White-Box Prediction of Process Performance Indicators via Flow Analysis

Ilya Verenich*
Queensland University of Technology
Brisbane, Australia
ilya.verenich@qut.edu.au

Hoang Nguyen
Queensland University of Technology
Brisbane, Australia
huanghuy.nguyen@hdr.qut.edu.au

Marcello La Rosa
Queensland University of Technology
Brisbane, Australia
m.larosa@qut.edu.au

Marlon Dumas†
University of Tartu
Tartu, Estonia
marlon.dumas@ut.ee

ABSTRACT

Predictive business process monitoring methods exploit historical process execution logs to provide predictions about running instances of a process, which enable process workers and managers to preempt performance issues or compliance violations. A number of approaches have been proposed to predict quantitative process performance indicators, such as remaining cycle time, cost, or probability of deadline violation. However, these approaches adopt a black-box approach, insofar as they predict a single scalar value without decomposing this prediction into more elementary components. In this paper, we propose a white-box approach to predict performance indicators of running process instances. The key idea is to first predict the performance indicator at the level of activities, and then to aggregate these predictions at the level of a process instance by means of flow analysis techniques. The paper specifically develops this idea in the context of predicting the remaining cycle time of ongoing process instances. The proposed approach has been evaluated on four real-life event logs and compared against several baselines.

CCS CONCEPTS

•**Information systems** → *Information systems applications; Decision support systems;*

KEYWORDS

Process Mining, Predictive Process Monitoring, Flow analysis

ACM Reference format:
Ilya Verenich, Hoang Nguyen, Marcello La Rosa, and Marlon Dumas. 2017. White-Box Prediction of Process Performance Indicators via Flow Analysis.

*This author is also affiliated with the Institute of Computer Science, University of Tartu, Estonia.
†This author is also affiliated with the Queensland University of Technology, Australia.

In *Proceedings of 2017 International Conference on Software and Systems Process , Paris, France, July 2017 (ICSSP'17),* 10 pages.
DOI: 10.1145/3084100.3084110

1 INTRODUCTION

Predictive business process monitoring techniques seek to determine the future state or properties of ongoing process instances based on models extracted from historical event logs. A wide range of predictive monitoring techniques have been proposed to predict for example compliance violations [13, 14], the next activity or the remaining sequence of activities of a process instance [8, 23], or quantitative process performance indicators, such the remaining cycle time of a process instance [18, 19, 22]. These predictions can be used to alert process workers to problematic process instances or to support resource allocation decisions, e.g. to allocate additional resources to instances that are at risk of a deadline violation.

This paper addresses the problem of predicting quantitative process performance indicators, with a specific focus on predicting the remaining cycle time of ongoing process instances. Existing approaches to this problem adopt a "black-box" approach by building stochastic models or regression models which, given a process instance, predict the remaining execution time as a single scalar value, without seeking to explain this prediction in terms of more elementary components. Yet, quantitative performance indicators such as cost or time are aggregations of corresponding performance indicators of the activities composing the process. In particular, the cycle time of a process instance consists of the sum of the cycle time of the activities performed in that process instance. In this respect, existing techniques allow us to predict the aggregate value of a performance indicator for a running process instance, but they do not explain how each activity contributes to this aggregate prediction.

Motivated by this observation, this paper proposes a "white-box" approach to predicting quantitative performance indicators of running process instances based on a general technique for quantitative process analysis known as flow analysis. The idea of flow analysis is to estimate a quantitative performance indicator at the level of a process by aggregating the estimated values of this performance indicator at the level of the activities in the process, taking into account the control-flow relations between these activities. Accordingly, in order to predict the remaining cycle time of a process instance, we propose to first estimate the cycle time of each activity

that might potentially be executed within this process instance, and then to aggregate these estimates using flow analysis.

In addition to providing predictions that can be traced down to the level of individual activities, we show via an empirical evaluation with real-life business process event logs, that the proposed technique achieves comparable and sometimes higher prediction accuracy relative to several state-of-the-art "black-box" baselines.

The remainder of the paper is structured as follows. Section 2 presents the related work on process prediction, with an emphasis on the prediction of remaining time. Section 3 introduces the important concepts and notations used in the paper. Section 4 outlines the details of the proposed approach. Next, Section 5 presents an experimental evaluation of our approach and compares it with the baseline techniques. Finally, Section 6 concludes the paper and outlines future work directions.

2 RELATED WORK

A wide range of predictive business process monitoring problems have been studied in previous work, including the prediction of delays and deadline violations, remaining cycle time, outcome, and future events of a running case.

The problem of predicting delays and deadline violations in business processes has been addressed by different authors. Pika et al. [16] propose a technique for predicting deadline violations by identifying process risk indicators that cause the possibility of a delay. Metzger et al. [15] present techniques for predicting "late show" events (i.e. delays between the expected and the actual time of arrival) in a freight transportation process by finding correlations between "late show" events and external variables related to weather conditions or road traffic. Finally, Senderovich et al. [20] apply queue mining techniques to predict delays in case executions.

Another group of works address the prediction of the remaining cycle time of running cases. Van Dongen et al. predict the remaining time by fitting non-parametric regression models based on the frequencies of activities within each case, their average durations, and case attributes [24]. Van der Aalst et al. [22] propose a remaining time prediction method by constructing a transition system from the event log using set, bag, or sequence abstractions of observed events. Polato et al. [17] refine this method by proposing a data-aware transition system annotated with classifiers and regressors. Rogge-Solti and Weske [18, 19] model business processes as stochastic Petri nets and perform Monte Carlo simulation to predict the remaining time of a process instance. De Leoni et al. [5, 6] propose a general framework to predict various characteristics of running instances, including the remaining time, based on correlations with other characteristics and using decision and regression trees. The remaining time prediction problem has also been extensively studied in the context of software development processes. For example, Kikas et al. [10] predict issue resolution time in Github projects using static, dynamic and contextual features. In this paper, we show that the remaining cycle time of a process instance can be decomposed into a sum of the cycle times of the activities that are yet to be performed in that process instance. Thus, estimating cycle times of individual activities, we can estimate the entire remaining time of a case.

Another category of techniques aim to predict the outcome of running cases. For example, Maggi et al. [14], propose a framework to predict the outcome of a case (normal vs. deviant) based on the sequence of activities executed in a given case and the values of data attributes of the last executed activity in a case. This latter framework constructs a classifier on-the-fly (e.g. a decision tree or random forest) based on historical cases that are similar to the (incomplete) trace of a running case. Other approaches construct a collection of classifiers offline. For example, [13] construct one classifier for every possible prediction point (e.g. predicting the outcome after the first event, the second one and so on). Conforti et al [4] apply a multi-classifier (decision trees) at each decision point of the process, to predict the likelihood of various types of risks, such as cost overruns and deadline violations.

A final group of techniques aim to predict future event(s) of a running case. Lakshmanan et al. [12] use Markov chains to estimate the probability of future execution of a given task in a running case; Breuker et al. [3] use probabilistic finite automata to predict the next activity to be performed while Tax et al [21] predict the entire continuation of a running case as well as timestamps of future events using long short-term memory (LSTM) neural networks.

In this paper, we do not address the problems of case outcome prediction and future events prediction, although our approach could in principle be extended in these directions.

3 BACKGROUND

In this section, we introduce concepts used in later sections of this paper.

3.1 Event logs, traces and sequences

For a given set A, A^* denotes the set of all sequences over A and $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$ a sequence of length n; $\langle\rangle$ is the empty sequence and $\sigma_1 \cdot \sigma_2$ is the concatenation of sequences σ_1 and σ_2. $hd^k(\sigma) = \langle a_1, a_2, \ldots, a_k \rangle$ is the prefix of length k ($0 < k < n$) of sequence σ and $tl^k(\sigma) = \langle a_{k+1}, \ldots, a_n \rangle$ is its suffix. For example, for a sequence $\sigma_1 = \langle a, b, c, d, e \rangle$, $hd^2(\sigma_1) = \langle a, b \rangle$ and $tl^2(\sigma_1) = \langle c, d, e \rangle$.

Let $\mathcal{E}$ be the event universe, i.e., the set of all possible event identifiers, and $\mathcal{T}$ the time domain. We assume that events are characterized by various properties. One of these properties is the timestamp of an event[1], meaning that there is a function $\pi_{\mathcal{T}} \in \mathcal{E} \rightarrow \mathcal{T}$ that assigns timestamps to events. Other properties of an event include its activity, resource performing the event, etc.

Definition 3.1 (Trace). A *trace* is a finite non-empty sequence of events $\sigma \in \mathcal{E}^*$ such that each event appears only once and time is non-decreasing, i.e., for $1 \leq i < j \leq |\sigma| : \sigma(i) \neq \sigma(j)$ and $\pi_{\mathcal{T}}(\sigma(i)) \leq \pi_{\mathcal{T}}(\sigma(j))$. A trace in a log represents the execution of one case.

Definition 3.2 (Event log). An *event log* is a set of events, each linked to a particular trace and globally unique, i.e., the same event cannot occur twice in a log.

3.2 Flow analysis

Flow analysis is a family of techniques that enables estimation of the overall performance of a process given knowledge about the

[1] Hereinafter, we refer to the event *completion* timestamp unless otherwise noted.

performance of its activities. For example, using flow analysis one can calculate the average cycle time of an entire process if the average cycle time of each activity is known. Flow analysis can also be used to calculate the average cost of a process instance knowing the cost-per-execution of each activity, or calculate the error rate of a process given the error rate of each activity [7]. The main advantage of the flow analysis is that the estimation can be easily explained in terms of its elementary components.

Definition 3.3 (Cycle time of an activity). A *cycle time* of an activity i is the average time it takes between the moment the activity is ready to be executed and the moment it completes. By "ready to be executed" we mean that all activities upon which the activity in question depends have completed. Formally, cycle time is the difference between the timestamp of the activity and the timestamp of the previous activity. i.e. $\pi_{\mathcal{T}}(\sigma(i)) - \pi_{\mathcal{T}}(\sigma(i-1))$ for $1 \leq i \leq |\sigma|$. Here, $\pi_{\mathcal{T}}(\sigma(0))$ denotes the start time of the case.

The cycle time of an activity includes the processing time of the activity, as well as all waiting time prior to the execution of the activity. Processing time refers to the time that actors spend doing actual work. On the other hand, waiting time is the portion of the cycle time where no work is being done to advance the process. This may include time spent in transferring information about the case between process participants, for example when documents are exchanged by post, as well as time when the case is waiting for an actor to process it. In many processes, the waiting time makes up a considerable proportion of the overall cycle time. This situation may, for example, happen when the work is performed in batches. In a process related to the approval of purchase requisitions at a company, the supervisor responsible for such approvals in a business unit might choose to batch all applications and check them only once at the start or the end of a working day [7].

To understand how flow analysis works, we start with an example of a process with *sequential* fragments of events as in Figure 1a. Each fragment has a single entry flow and a single exit flow and has a cycle time T_i. Since the fragments are performed one after the other, we can intuitively conclude that the cycle time CT of a purely sequential process with N event fragments is the sum of the cycle times of each fragment [7]:

$$CT = \sum_{i=1}^{N} T_i \tag{1}$$

Let us consider a process model with a decision point between N mutually exclusive fragments, represented by an *XOR gateway* (Figure 1b). In this case, the cycle time of a process model is

$$CT = \sum_{i=1}^{N} p_i \cdot T_i, \tag{2}$$

where p_i denote the branching probabilities, i.e. frequencies with which a given branch i of a decision gateway is taken.

In case of parallel, or *AND gateways* where activities can be executed concurrently as in Figure 1c, the combined cycle time of multiple fragments is determined by the slowest of the fragments, that is:

$$CT = \max_{i=1\ldots n} T_i \tag{3}$$

Another recurrent pattern is the one where a fragment of a process may be repeated multiple times, for instance because of a failed quality control. This situation is called *rework* and is illustrated in Figure 1d. The fragment is executed once. Next, it might be repeated each time with a probability r referred to as *rework probability*. The average number of times that the rework fragment is expected to be executed can be obtained via the geometric series [7], and the cycle time of the fragment in this case is:

$$CT = \frac{T}{1-r} \tag{4}$$

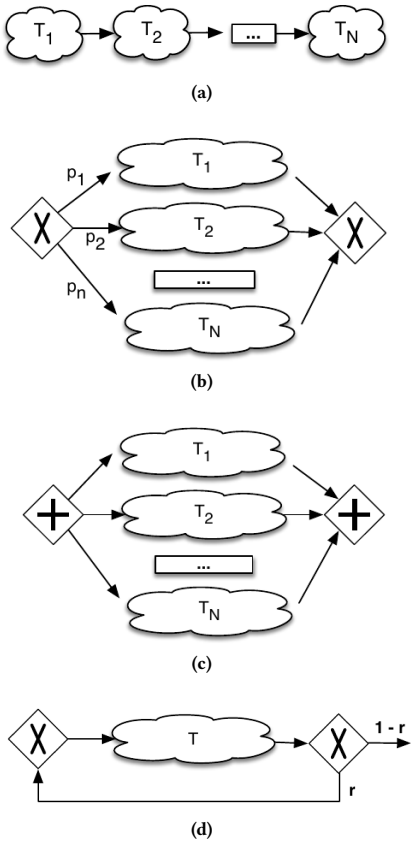

Figure 1: Typical process model patterns: sequential (a), XOR-block (b), AND-block (c) and rework loop (d).

Besides cycle time, flow analysis can also be used to calculate other performance measures. For instance, assuming we know the average cost of each activity, we can calculate the cost of a process more or less in the same way as we calculate cycle time. In particular, the cost of a sequence of activities is the sum of the costs of these activities. The only difference between calculating cycle time and calculating cost relates to the treatment of AND-blocks. The cost of an AND-block such as the one shown in Figure 1c is not the maximum of the cost of the branches of the AND-block. Instead, the cost of such a block is the sum of the costs of the branches. This is because after the AND-split is traversed, every branch in the AND join is executed and therefore the costs of these branches add up to one another [7].

In case of block-structured process models that can be represented as a sequence of event fragments with a single entry and a single exit, we can relate each fragment to one of the four described types and use the aforementioned equations to estimate the required performance measure. However, in case of an unstructured process model or if a model contains other modeling constructs besides AND and XOR gateways, the method for calculating performance measures becomes more complicated.

A major limitation of flow analysis is that it does not consider the fact that a process behaves differently depending on the load, i.e. the number of process instances that are running concurrently. For example, the cycle time of a process for handling insurance claims would be much slower if the insurance company was handling thousands of claims at once, due for example to a recent natural disaster as compared to the case where the load is low and the company may be handling only a hundred claims at once. When the load increases and the number of process workers remains constant, the waiting times tend to increase. This phenomenon is referred to as *resource contention*. It occurs when there is more work to be done than resources available to perform the work. In such scenarios, some tasks will be in waiting mode until a required resource is freed up. Flow analysis does not take into account the effects of increased resource contention. Instead, the estimates obtained from flow analysis are only applicable if the level of resource contention is relatively stable over the long term.

4 APPROACH

In this section, we describe the proposed approach to predict the remaining time. We first provide an overview of the entire solution framework and then focus on the key parts of our approach.

4.1 Overview

Our approach exploits historical execution traces in order to discover a structured process model. Once the model has been discovered, we identify its set of activities and decision points and train two families of machine learning models: one to predict the cycle time of each activity, and the other to predict the branching probabilities of each decision point. To speed up the performance at runtime, these steps are performed offline (Figure 2).

At runtime, given an ongoing process instance, we align its partial trace with the discovered process model to determine the current state of the instance. Next, we traverse the process tree obtained from the model starting from the state up to the process end and deduce a formula for remaining time using rules described in Section 3.2. The formula includes cycle times of activities and branching probabilities of decision points that are reachable from the current execution state. These components are predicted using previously trained regression and classification models. Finally, we evaluate the formula and obtain the expected value of the remaining cycle time.

4.2 Discovering Process Models From Event Logs

The proposed approach relies on a process model as input. However, since the model is not always known or might not conform to the real process, generally we need to discover the model from

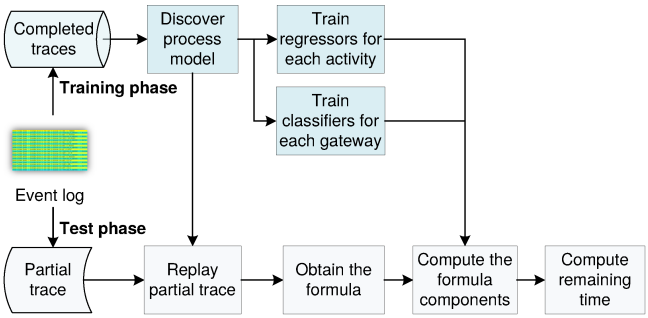

Figure 2: Overview of the proposed approach.

event logs. For that, we use a two-step automated process discovery technique proposed in [2] that has been shown to outperform traditional approaches with respect to a range of accuracy and complexity measures. The technique has been implemented as a standalone tool [1] as well as a ProM plugin, namely *StructuredMiner*.

The technique in [2] pursues a two-phase "discover and structure" approach. In the first phase, a model is discovered from the log using a heuristic process discovery method that has been shown to consistently produce accurate, but potentially unstructured or even unsound models. In the second phase, the discovered model is transformed into a sound and structured model by applying two techniques: a technique to maximally block-structure an acyclic process model and an extended version of a technique for block-structuring flowcharts. This approach has been shown to outperform traditional "discover structured" approaches with respect to a range of accuracy and complexity measures.

A structured model is internally represented as a *process tree*. A process tree is a tree where each leaf is labeled with an activity and each internal node is labeled with a control-flow operator: *sequence*, *exclusive choice*, *non-exclusive choice*, *parallelism*, or *iteration*.

4.3 Replaying Partial Traces on the Process Model

For a given partial trace, to predict its remaining time, we need to determine the current state of the trace relative to the process model. For that, we map, or align, a trace to the process model using the technique described in [1] which is available as a plugin for the open-source process mining platform Apromore.

The technique treats a process model as a graph that is composed of activities as nodes and their order dependencies as arcs. A case replay can be seen as a series of coordinated *moves*, including those over the model activities and gateways and those over the trace events. In that sense, a case replay is also termed an *alignment* of a process model and a trace. Ideally, this alignment should result in as many matches between activity labels on the model and event labels in the trace as possible. However, practically, the replay may choose to skip a number of activities or events in search of more matches in later moves. Moves on the model must observe the semantics of the underlying modeling language which is usually expressed by the notion of *tokens*. For example, for a BPMN model, a move of an incoming token over a XOR split gateway will result in a single token produced on one of the gateway outgoing branches,

[1] Available at http://apromore.org/platform/tools

while a move over an AND split gateway will result in a separate token produced on each of the gateway outgoing branches. The set of tokens located on a process model at a point in time is called a *marking*. On the other hand, a move in the trace is sequential over successive events of the trace ordered by timestamps, one after another. Thus, after every move, either on the model or in the trace, the alignment comes to a *state* consisting of the current marking of the model and the index of the current event in the trace.

In [1], cases are replayed using a heuristics-based backtracking algorithm that searches for the best alignment between the model and a partial trace. The algorithm can be illustrated by a traversal of a process tree starting from the root node, e.g. using depth-first search, where nodes represent partial candidate solution states (Figure 3). Here the state represents the aforementioned alignment state of the case replay. At each node, the algorithm checks whether the alignment state till that node is good enough. If so, it generates a set of child nodes of that node and continues down that path; otherwise, it stops at that node, i.e. it prunes the branch under the node, and backtracks to the parent node to traverse other branches.

procedure EXPLORE(node n)
  if REJECT(n) **then return**
  if COMPLETE(n) **then**
    OUTPUT(n)
  for n_i : CHILDREN(n) **do** EXPLORE(n_i)

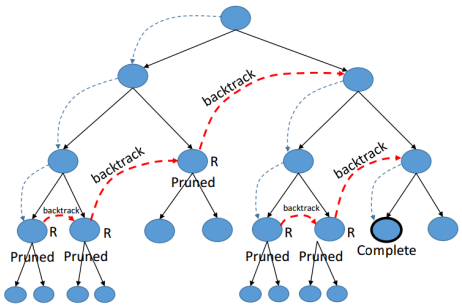

Figure 3: Backtracking algorithm (taken from [1]).

4.4 Obtaining the flow analysis formulas

Having determined the current state of the case execution, we traverse the process model starting from that state until the process completion in order to obtain the flow analysis formulas.

As a running example, let us consider a simple process model in Figure 4. Applying the flow analysis formulas described in Section 3.2, the average cycle time of this process can be decomposed as follows:

$$CT = T_A + \max(T_B + T_C, T_D) + T_F + p2\left(T_G + \frac{T_H}{1-r}\right) \quad (5)$$

Note that one of the branches of gateway $X21$ is empty and therefore does not contribute to the cycle time. Therefore, only the branch with the probability $p2$ is included in the equation.

The components of the formula – cycle times of individual activities and branching and rework probabilities – can be estimated as averages of their historical values. However, since we deal with *ongoing* process cases, we can use the information that is already available from the case prefix to *predict* the above components.

Consider, we have a partial trace $hd(\sigma) = \langle A, D, B\rangle$. Replaying this trace on the given model as described in the Section 4.3, we find the current marking to be in the states B and D within the AND-block. Traversing the process model starting from these states until the process end, we obtain the following formula:

$$CT_{rem} = \max(\mathbf{T_B} + T_C, \mathbf{T_D}) + T_F + p2\left(T_G + \frac{T_H}{1-r}\right) \quad (6)$$

Since the activity A has already been executed, it does not contribute to the *remaining* cycle time. Thus, it is not a part of the formula. Furthermore, $\mathbf{T_B}$ and $\mathbf{T_D}$ have been executed, however, since they form one of the terms of the formula wherein T_C is still unknown, they cannot be omitted, but their *actual* cycle times should be taken. All the other formula terms need to be predicted using the data from $hd(\sigma)$.

Similarly, if a current marking is inside a XOR block, its branching probabilities need not be predicted. Instead, the probability of the branch that has actually been taken is set to 1 while the other probabilities are set to 0.

A more complex situation arises when the current marking is inside the rework loop. In this case, we "unfold" the loop as shown in the Figure 5. Specifically, we separate the already executed occurrences of the rework fragment from the potential future occurrences and take the former out of the loop. Let us consider a partial trace $hd(\sigma) = \langle A, D, B, C, F, G, H\rangle$. Since H has occurred once, according to the process model (Figure 4), with a probability r, it may be repeated, otherwise, the rework loop is exited. To signal this choice, we take the first occurrence of H out of the loop, and place a XOR gateway after it. One of the branches will contain a rework loop of future events with the same probability r, while the other one will reflect an option to skip the loop altogether. Thus, the cycle time of the whole fragment can be decomposed as follows:

$$CT_H = \mathbf{T_{H'}} + r\frac{T_H}{1-r}, \quad (7)$$

where $\mathbf{T_{H'}}$ refers to the cycle time of already executed occurrence(s) of H. It is highlighted in bold font, meaning that we should take the actual cycle time rather than the predicted.

4.5 Computing the remaining time

We can use the flow analysis formulas produced by the method described in Section 4.4 to compute the remaining cycle time of a case, given: (i) an estimate of the cycle time of each activity reachable from the current execution state; and (ii) an estimate of the branching probability of each flow stemming from a reachable XOR-split (herein called a reachable *conditional flow*). Given an execution state, these estimates can be obtained in several ways including:

(1) By using the prediction models produced for each reachable activity and for each reachable conditional flow, taking into account only traces that reach the current execution state. We herein call this approach *predictive flow analysis*.
(2) By computing the mean cycle time of each reachable activity and the traversal frequency of each reachable conditional flow, again based only on the suffixes of traces that reach the execution state in question. We call this approach *mean flow analysis*

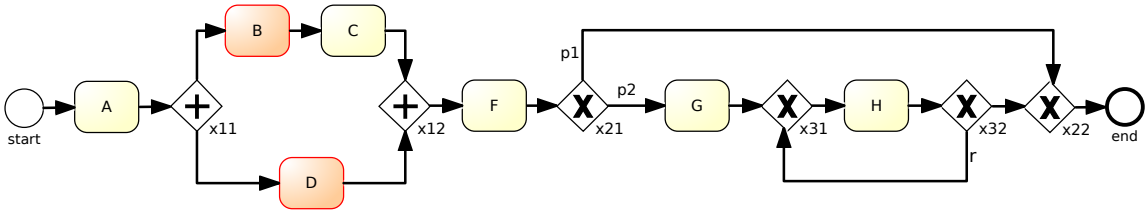

Figure 4: Example process model. Highlighted is the current marking

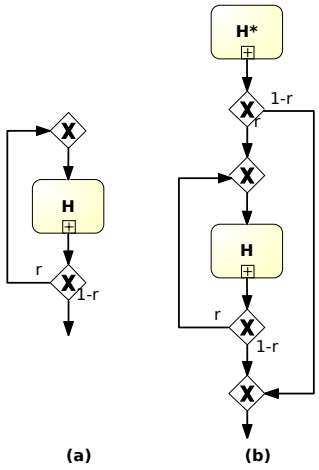

Figure 5: Unfolding the rework loop of F

The rationale for the mean flow analysis is that the prefix size can have two opposite effects on prediction accuracy. If a prefix is too short, there might not be enough information in it to predict cycle times of some activities and gateways' branching probabilities, especially those that are executed near the process end. On the other hand, if the prefix is long, for activities and gateways that are usually executed at the beginning of the process, we will not have enough training data to fit the model. As an example, let us consider an activity that, according to the process model, usually occurs in the 4th or 5th position in the process, but in a few cases can occur in the 8th position. Then, to fit a model for a prefix length 5, as training data we can only use these few cases, since for most other cases, the activity will not occur after the 5th event. In cases where the accuracy of the produced predictive models is insufficient, we can then use the mean historical activity cycle times instead.

In order to make use of predictive models, we need to encode process execution traces in the form of feature vectors. In this paper, we use *index-based encoding* as described in [13] that concatenates the case attributes and, for each position in a trace, the event occurring in that position and the value of each event attribute in that position. This type of trace encoding is lossless and has been shown to achieve a relatively high accuracy and reliability when making early predictions of binary process properties [13, 25].

For each activity in the process model, to predict its cycle time, we train a regression model, while for predicting branching probabilities we fit classification models for each corresponding XOR gateway. In the latter case, each branch of a gateway is assigned a class starting from 0, and the model makes predictions about the probability of each class. The predictive models are trained for prefixes $hd^k(\sigma)$ of all traces σ in the training set for $2 \leq k < |\sigma|$. We do not train and make predictions after the first event, since for those prefixes there is no sufficient data available to base the predictions upon.

As an example, let us consider a snapshot of the log with one completed case in Table 1 that corresponds to the process model in Figure 4. The events are ordered according to their completion timestamp.

Table 1: Extract of an event log.

Case	Case attributes		Event attributes		
ID	Channel	Age	Activity	Timestamp	Resource
1	Email	37	A	9:13:00	R03
1	Email	37	B	9:14:20	R12
1	Email	37	D	9:16:00	R07
1	Email	37	C	9:18:00	R03
1	Email	37	F	9:18:10	R21
1	Email	37	G	9:18:50	R12
1	Email	37	H	9:19:00	R12

To encode traces as feature vectors, we include both case attributes and event attributes. Thus, the first case in the log will be encoded as such:

$$\vec{X} = (\text{Email}, 37; \text{A}, \text{B}, \text{D}, \text{C}, \text{F}, \text{G}, \text{H}; 9{:}13{:}00, \text{R03}; 9{:}14{:}20, \text{R12}; 9{:}16{:}00, \text{R07}; 9{:}18{:}00, \text{R03}; 9{:}18{:}10, \text{R21}; 9{:}18{:}50, \text{R12}; 9{:}19{:}00, \text{R12})$$

Now, to create the training set for $hd^k(\sigma)$, we cut the feature vectors to include the event attributes up to the k-th event and case attributes (which are usually known since the beginning of the case). Furthermore, we add the value of the target variable y to be learned. For example, if we are to predict the cycle time of activity G for prefixes $k = 2$, the training sample based on the data extracted from the first case in Table 1 would be created as follows:

$$D_G^2 = \left\{\vec{X^2}, y_G\right\} = \{\text{Email}, 37; \text{A}, \text{B}; 9{:}13{:}00, \text{R03}; 9{:}14{:}20, \text{R12};\ 40\}$$

Here 40 is the cycle time of G for the first case, determined as the time difference (in seconds) between the completion timestamp of G and the completion timestamp of the previous activity F. It should be noted that for a case that follows the upper branch of the gateway $x21$, the process terminates after F, thus G is never executed and its cycle time is undefined. Therefore, we exclude such cases from the training data. Conversely, if an activity occurs multiple times in a case, we take its average cycle time.

Similarly, if we are to predict the branching probabilities for $X32$ gateway for prefixes $k = 2$, we would assign class 0 to the branch that leads to rework and class 1 to the other branch. Then, the first training sample would be:

$$D^2_{x32} = \left\{\vec{X^2}, y_{x32}\right\} = \{\text{Email}, 37; \text{A}, \text{B}; 9{:}13{:}00, \text{R03}; 9{:}14{:}20, \text{R12};\ 1\}$$

Since H is not repeated for the first case, we assign class 1 to the gateway. Evidently, the probability of class 0 would be equal to the rework probability r.

5 EVALUATION

In the following section, we empirically compare the predictive flow analysis and the mean approaches between them and against baselines proposed in previous work. In particular, we seek to answer the following specific research questions:

RQ1. Do flow analysis-based techniques provide *accurate* predictions in comparison with state-of-the-art baselines?

RQ2. Do flow analysis-based techniques provide *stable* results at different stages of ongoing cases?

The first question focuses on the quality of the predictions, while the second one relates to the stability of the results at different stages of running cases. Next, we describe the conducted experiments to answer these research questions. The source code and supplementary material required to reproduce the experiments reported in this paper can be found at http://github.com/verenich/flow-analysis-predictions

5.1 Datasets

We conducted the experiments using four real-life event datasets. Table 2 summarizes the basic characteristics of each dataset.

First three datasets originate from the Business Process Intelligence Challenge (BPIC'12)[1] and contain data from the application procedure for financial products at a large financial institution. This process consists of three subprocesses: one that tracks the state of the application (BPIC'12 A), one that tracks the state of the offer (BPIC'12 O), and a third one tracks the states of work items associated with the application (BPIC'12 W). For the latter subprocess, we retain only events of type *complete*. The fourth dataset is based on the log that contains events from a ticketing management process of the help desk of an Italian software company[2]. Each case starts with the insertion of a new ticket into the ticketing management system and ends when the issue is resolved and the ticket is closed.

As mentioned in Section 3.2, flow analysis technique cannot readily deal with unstructured models. Even though the tool described in Section 4.2 aims to mine maximally structured models, it does not always succeed in doing so. Specifically, it sometimes produces models with overlapping loops which our current implementation is unable to deal with. One solution to this problem could be to simplify the process model by removing the transitions that cause overlapping loops. However, this may severely decrease the accuracy of the discovered model, which will, in turn, negatively affect the accuracy of the flow analysis-based predictions of remaining time. Hence, instead, we remove the cases that cause overlapping loops from the event log (up to 15% of cases in each log).

Table 2: Summary of datasets.

Dataset	Number of			Mean	Mean case
	cases	activities	case variants	events per case	duration, days
BPIC'12 A	12,007	10	10	4.49	7.5
BPIC'12 O	3,487	7	6	4.56	15.1
BPIC'12 W	9,650	6	2,263	7.50	11.4
helpdesk	3,218	5	8	3.30	7.3

5.2 Experimental setup

To assess the quality of the prediction of continuous variables, well-known error metrics are Mean Absolute Error (MAE), Root Mean Square Error (RMSE) and Mean Percentage Error (MAPE) [9], where MAE is defined as the arithmetic mean of the prediction errors, RMSE – as the square root of the squared prediction errors, while MAPE measures error as the average of the unsigned percentage error. We observe that the value of remaining time tends to be highly varying across cases, with values at different orders of magnitude. RMSE would be very sensitive to such outliers. Furthermore, the remaining time can be very close to zero, especially near the end of the trace, thus MAPE is skewed in such situations. Hence, we use MAE to measure the error in predicting the remaining time.

We employ several baselines to compare our approach to. Firstly, we use a transition system (TS) based method proposed by van der Aalst et al. [22] applying both set, bag and sequence abstractions. Secondly, we use a method proposed by Leontjeva et al. [13] who compared several types of business process sequence encodings for prediction of the boolean case outcome. This method can be naturally adjusted to predict the remaining time by replacing the classification task with the regression task. For the purpose of this paper, we will reproduce only two types of the original encodings – index-based and frequency-based, as the others were shown to have either very similar or inferior performance. Next, we evaluate against the stochastic Petri-net (SPN) based approach proposed by Rogge-Solti and Weske [18, 19]. Specifically, we use the method based on the *constrained* Petri net, as it was shown to have the lowest prediction error. However, their original approach makes predictions at fixed time points, regardless of the arriving events. To make the results comparable to our approach, we modify the method to make predictions after each arrived event. Finally, we used a combined estimator along the lines of [24] where the feature set includes the frequencies of activities within each case, their average durations, and case attributes.

In our experiments, we order the cases in the logs based on the time at which the first event of each case has occurred. Then, we split the logs into two parts. We use the first part (2/3 of the cases) as a training set, i.e. as historical data to train the predictive models. The remaining 1/3 of the cases are used to evaluate the accuracy of the predictions. Furthermore, we perform a five-fold cross-validation on the training set in order to select the optimal values of the training parameters such as the number of trees and the number of variables at each split for a random forest model.

[1] doi:10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
[2] doi:10.17632/39bp3vv62t.1

5.3 Results

Table 3 summarizes the performance of the predictive and mean flow analysis techniques, as well as the baselines approaches for each dataset. We make predictions for prefixes $hd^k(\sigma)$ of traces σ in the test set starting from $k = 2$. However, since for very long prefixes, there are not enough traces with that length, and the error measurements become unreliable, we stop the predictions after k reaches the 70th-percentile length of the traces in the log, i.e. at least 70% of the traces in the log have a length smaller than k. Thus, since the BPIC'12 W log contains longer traces, the prefix sizes evaluated are higher for this log. Additionally, we report the average performance across all prefixes, weighted over the relative frequency of traces with that prefix (i.e. longer prefixes get lower weights, since not all traces reach that length).

We observe that for most logs, the prediction accuracy of flow analysis-based techniques is at least as good as that of the baselines. At the same time, for all logs except BPIC'12 O, mean flow analysis, on average, provides the best results among all the methods. Specifically, it outperforms the predictive flow analysis. The latter is due to the lack of data attributes in the event logs that would be able to accurately explain the variation in the cycle times of individual activities and branching probabilities of each conditional flow. To further investigate this issue, for each activity in the BPIC'12 A and BPIC'12 O logs, we analyze the performance of regressors trained to predict its cycle time and compare it with a constant regressor used in the mean flow analysis. In Table 4 we report MAE of cycle times for each activity and each technique, as evaluated on the test set. Since for each prefix length we have a separate regressor, we report *weighted* average values, as in Table 3. In addition, we report the actual average cycle time values of each predicted activity based on the test set.

As can be seen from Table 4, in the BPIC'12 O log, prediction-based cycle times are more accurate than the constant ones for longer activities which make up the largest portion of the remaining cycle time. Furthermore, the difference between the two approaches is higher for BPIC'12 O. Hence, for this log, we can estimate the remaining time more accurately with the predictive flow analysis.

Another observation is related to the very low accuracy of the predictive flow analysis on the BPIC'12 W log. Having closely inspected this log, we found that it contains sequences of two or more events in a row of the same activity. In other words, activities are frequently reworked multiple times. As mentioned in Section 3.2, flow analysis techniques assume a *constant* rework probability r. However, in many real-life processes r subsequently decreases after each execution of the rework loop, meaning that the rework becomes less and less likely. Thus, if r is inaccurately predicted in predictive flow analysis, this error propagates further. To verify our hypothesis, we modify the log keeping only the first occurrence of each repeated event in a sequence. To keep the remaining time calculations correct, we retain the last event of a case, even if it is a repeated event. Having run the experiments on the modified log (Table 5), we notice that predictive flow analysis becomes almost as accurate as mean flow analysis, thus proving our hypothesis.

Summing up, the experiments suggest that flow analysis-based techniques provide relatively accurate estimations of the remaining cycle time across all logs. Thus, we can positively answer **RQ1**.

Our experiments also show that flow analysis-based techniques are able to provide relatively accurate predictions starting from the early stages of an ongoing case. The general trend is a stable reduction in MAE values as a case progresses. This is due to the increasing amount of attributes in the prefix to base the predictions upon. Furthermore, the actual remaining times intuitively decreases at later stages of a case, thus its prediction error also decreases. We can then provide a positive answer to **RQ2**.

Execution Times. The execution time of the proposed approach is composed of the execution times of the following components: (i) training the predictive models; (ii) replaying the partial traces on the process model (finding an alignment) and deriving the formulas; (iii) applying the models to predict the cycle times and branching probabilities and calculating the overall remaining time. For real-time prediction, it is crucial to output the results faster than the mean case arrival rate. Thus, we also measured the average runtime overhead of our approach. All experiments were conducted on a laptop with a 2.4 GHz Intel Core i5 processor and 8 Gb of RAM.

For a given prefix length k, training all the models takes between 20 and 200 seconds depending on the prefix size and the number of models to train. Replaying the test traces takes between 5 and 45 seconds, for a given length of the prefix. Finally, making the predictions takes less than 4 seconds per prefix length. This shows that our approach performs within reasonable bounds for most online applications.

5.4 Threats to Validity

The datasets used in this evaluation, except for the BPIC'12 W, have only the completion timestamps, but not the start timestamps. Thus, it is impossible to discern the actual processing time from the waiting time. The latter can have a significant impact on the overall cycle time depending on the case arrival rate and the resource load. As these factors are not accounted for in the predictive models, their accuracy is rather low.

We reported the results with a single learning algorithm (random forest). With decision trees and gradient boosting, we obtained qualitatively the same results, relatively to the baselines. However, our approach is independent of the learning algorithm used. Thus, using a different algorithm does not in principle invalidate the results. That said, we acknowledge that the goodness of fit, as in any machine learning problem, depends on the particular classifier/regressor algorithm employed. Hence, it is important to test multiple algorithms for a given dataset, and to apply hyperparameter tuning, in order to choose the most adequate algorithm with the best configuration.

The proposed approach relies on the accuracy of the branching probability estimates provided by the classification model. It is known however that the likelihood probabilities produced by classification methods are not always reliable. Methods for estimating the reliability of such likelihood probabilities have been proposed in the machine learning literature [11]. A possible enhancement of the proposed approach would be to integrate heuristics that take into account such reliability estimates.

Table 3: MAE values (in days) for prefixes of different lengths.

Method	Prefix length									
	Avg	2	3	4	5	6	7	8	9	10
BPIC'12 A										
Predictive flow analysis	9.48	9.60	10.38	9.68	7.04					
Mean flow analysis	**8.32**	**7.89**	9.62	8.81	**6.88**					
TS set abstraction [22]	9.16	8.39	10.53	10.31	8.02					
TS bag abstraction [22]	9.16	8.39	10.53	10.31	8.02					
TS sequence abstraction [22]	9.16	8.39	10.53	10.31	8.02					
Index-based encoding [13]	9.07	8.21	10.48	10.38	7.99					
Frequency-based encoding [13]	9.16	8.40	10.52	10.28	8.02					
Constrained SPN [19]	8.44	9.15	**8.47**	**7.41**	6.89					
Combined estimator [24]	9.05	8.19	10.48	10.32	8.03					
BPIC'12 O										
Predictive flow analysis	5.96	7.46	**6.40**	2.55						
Mean flow analysis	6.33	8.00	6.81	2.53						
TS set abstraction [22]	6.05	8.03	6.81	2.54						
TS bag abstraction [22]	6.05	8.03	6.81	2.54						
TS sequence abstraction [22]	6.05	8.03	6.81	2.54						
Index-based encoding [13]	6.36	8.06	6.82	2.52						
Frequency-based encoding [13]	6.33	8.02	6.81	2.54						
Constrained SPN [19]	**5.49**	**6.46**	6.46	**2.42**						
Combined estimator [24]	6.34	8.04	6.80	2.52						
BPIC'12 W										
Predictive flow analysis	14.48	15.38	15.33	15.83	14.32	16.08	11.62	12.52	13.67	12.21
Mean flow analysis	**7.35**	**8.58**	**8.01**	**7.49**	**7.20**	**6.87**	**6.70**	**6.61**	**6.36**	**6.21**
TS set abstraction [22]	7.99	9.04	8.70	8.20	7.93	7.50	7.34	7.35	6.94	6.75
TS bag abstraction [22]	7.95	8.84	8.71	8.22	7.95	7.42	7.26	7.27	6.93	6.83
TS sequence abstraction [22]	7.91	8.84	8.70	8.22	7.91	7.40	7.21	7.22	6.84	6.74
Index-based encoding [13]	7.64	8.71	8.29	7.86	7.50	7.24	7.02	6.95	6.69	6.53
Frequency-based encoding [13]	7.79	8.77	8.64	8.19	7.93	7.40	7.20	7.24	6.85	6.66
Constrained SPN [19]	9.60	8.77	9.36	9.68	9.97	10.15	10.02	10.01	9.71	9.39
Combined estimator [24]	7.66	8.74	8.30	7.91	7.59	7.28	7.02	6.93	6.64	6.43
Helpdesk										
Predictive flow analysis	5.97	5.24	9.36	**2.76**						
Mean flow analysis	**5.27**	**5.10**	6.10	3.28						
TS set abstraction [22]	5.52	5.44	**5.92**	5.14						
TS bag abstraction [22]	5.59	5.49	6.15	3.08						
TS sequence abstraction [22]	5.59	5.49	6.15	3.08						
Index-based encoding [13]	5.58	5.39	6.54	3.26						
Frequency-based encoding [13]	5.61	5.50	6.17	3.28						
Constrained SPN [19]	5.54	5.34	6.53	4.29						
Combined estimator [24]	5.54	5.39	6.34	3.27						

6 CONCLUSION AND FUTURE WORK

The paper has put forward some potential benefits of a "white-box" approach to predicting quantitative process performance indicators. Rather than predicting single scalar indicators, we demonstrated how these indicators can be estimated as aggregations of corresponding performance indicators of the activities composing the process. In this way, the predicted indicators become more explainable, as they are decomposed into elementary components. Thus, business analysts can pinpoint the bottlenecks in the process execution and provide better recommendations to keep the process compliant with the performance standards.

We implemented and evaluated two approaches – one where the formulas' components are predicted from the trace prefix based on the models trained on historical completed traces, and the other one that instead uses constant values obtained from the historical averages of similar traces. We evaluated the approaches to predict the remaining cycle time, as one of common process performance indicators. The empirical evaluation has shown that the proposed techniques are, on average, able to yield more accurate predictions at different stages of running cases than the surveyed baselines.

We identified a limitation of flow analysis-based approaches when dealing with traces with rework loops, i.e. multiple occurrences of the same fragment of activities in a row. A direction for future work is to further investigate the factors affecting the performance of the proposed approaches in order to better understand their strength and weaknesses. Furthermore, we plan to extend

Table 4: MAE of cycle time predictions of individual activities and their actual mean cycle times (in days).

Activity	MAE		Mean cycle
	Predictive	Mean	time
BPIC'12 A			
A_CANCELLED	**11.97**	12.02	14.36
A_APPROVED	7.61	**7.51**	7.36
A_DECLINED	**3.72**	3.74	3.74
A_REGISTERED	**5.92**	5.96	3.70
A_ACTIVATED	**4.46**	4.47	2.88
A_ACCEPTED	**0.43**	0.78	0.76
A_PREACCEPTED	**0.04**	0.13	0.09
A_FINALIZED	0.01	**0.01**	0.01
BPIC'12 O			
O_CANCELLED	**8.68**	9.75	18.20
O_SENT_BACK	**2.79**	4.01	9.42
O_ACCEPTED	2.60	**2.59**	4.22
O_DECLINED	2.50	**2.43**	3.54
O_SENT	< 0.01	< 0.01	< 0.01

Table 5: MAE values (in days) for prefixes of different lengths for the modified *BPIC'12 W* log with excluded event duplicates.

Method	Prefix length				
	Avg	2	3	4	5
Predictive flow analysis	6.70	8.22	5.86	4.45	4.27
Mean flow analysis	**6.15**	**7.69**	**5.14**	3.88	4.14
TS Set abstraction [22]	6.70	8.40	5.82	3.94	4.27
TS Bag abstraction [22]	6.69	8.40	5.82	4.04	**3.97**
TS Sequence abstraction [22]	6.69	8.40	5.82	4.04	**3.97**
Index-based encoding [13]	6.54	8.14	5.66	4.15	4.04
Freq-based encoding [13]	6.71	8.44	5.83	4.03	4.00
Constrained SPN [19]	7.82	7.76	8.14	7.70	7.46
Combined estimator [24]	6.50	8.12	5.59	4.09	3.99

the proposed approaches so that they would be able to deal with more complex models with overlapping loops, using structuring techniques such as the one proposed in [26].

With some modifications in the derivation of the flow analysis formulas, the proposed approaches can be extended to predict other quantitative performance indicators. In future work, we aim to extend and evaluate the approaches to predict the process cost or error rate.

ACKNOWLEDGMENTS

This research is funded by the Australian Research Council under Grant No.: DP150103356 and the Estonian Research Council under Grant No.: IUT20-55.

REFERENCES

[1] Robert Andrews, Suriadi Suriadi, Moe Wynn, Arthur ter Hofstede, Nguyen Hoang Pika, Anastasiia, and Marcello la Rosa. 2016. Comparing static and dynamic aspects of patient flows via process model visualisations. *Preprint available at https://eprints.qut.edu.au/102848/* (2016).

[2] Adriano Augusto, Raffaele Conforti, Marlon Dumas, Marcello La Rosa, and Giorgio Bruno. 2016. Automated Discovery of Structured Process Models: Discover Structured vs. Discover and Structure. In *Conceptual Modeling - 35th International Conference, ER 2016*. 313–329.

[3] Dominic Breuker, Martin Matzner, Patrick Delfmann, and Jörg Becker. 2016. Comprehensible predictive models for business processes. *MIS Quarterly* 40, 4 (2016), 1009–1034.

[4] Raffaele Conforti, Massimiliano de Leoni, Marcello La Rosa, Wil M. P. van der Aalst, and Arthur H. M. ter Hofstede. 2015. A recommendation system for predicting risks across multiple business process instances. *Decision Support Systems* 69 (2015), 1–19.

[5] Massimiliano de Leoni, Wil M. P. van der Aalst, and Marcus Dees. 2014. A General Framework for Correlating Business Process Characteristics. In *BPM*. 250–266.

[6] Massimiliano de Leoni, Wil M. P. van der Aalst, and Marcus Dees. 2016. A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. *Information Systems* 56 (2016), 235–257.

[7] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. 2013. *Fundamentals of Business Process Management*. Springer.

[8] Joerg Evermann, Jana-Rebecca Rehse, and Peter Fettke. 2016. A Deep Learning Approach for Predicting Process Behaviour at Runtime. In *Proceedings of the 1st International Workshop on Runtime Analysis of Process-Aware Information Systems*. Springer.

[9] Rob J Hyndman and Anne B Koehler. 2006. Another look at measures of forecast accuracy. *International Journal of Forecasting* 22, 4 (2006), 679–688.

[10] Riivo Kikas, Marlon Dumas, and Dietmar Pfahl. 2016. Using dynamic and contextual features to predict issue lifetime in GitHub projects. In *Proceedings of the 13th International Conference on Mining Software Repositories, MSR*. 291–302. DOI: http://dx.doi.org/10.1145/2901739.2901751

[11] Meelis Kull and Peter A. Flach. 2014. Reliability Maps: A Tool to Enhance Probability Estimates and Improve Classification Accuracy. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2014*. 18–33.

[12] Geetika T Lakshmanan, Davood Shamsi, Yurdaer N Doganata, Merve Unuvar, and Rania Khalaf. 2015. A Markov prediction model for data-driven semi-structured business processes. *Knowledge and Information Systems* 42, 1 (2015), 97–126.

[13] Anna Leontjeva, Raffaele Conforti, Chiara Di Francescomarino, Marlon Dumas, and Fabrizio Maria Maggi. 2015. Complex Symbolic Sequence Encodings for Predictive Monitoring of Business Processes. In *BPM*. 297–313.

[14] Fabrizio Maria Maggi, Chiara Di Francescomarino, Marlon Dumas, and Chiara Ghidini. 2014. Predictive monitoring of business processes. In *CAiSE*. Springer, 457–472.

[15] Andreas Metzger, Rod Franklin, and Yagil Engel. 2012. Predictive monitoring of heterogeneous service-oriented business networks: The transport and logistics case. In *2012 Annual SRII Global Conference*. IEEE, 313–322.

[16] Anastasiia Pika, Wil M P van der Aalst, Colin J Fidge, Arthur H M ter Hofstede, and Moe T Wynn. 2012. Predicting deadline transgressions using event logs. In *BPM*. Springer, 211–216.

[17] Mirko Polato, Alessandro Sperduti, Andrea Burattin, and Massimiliano de Leoni. 2014. Data-aware remaining time prediction of business process instances. In *2014 International Joint Conference on Neural Networks, IJCNN 2014*. 816–823.

[18] Andreas Rogge-Solti and Mathias Weske. 2013. Prediction of remaining service execution time using stochastic Petri nets with arbitrary firing delays. In *ICSOC*. Springer, 389–403.

[19] Andreas Rogge-Solti and Mathias Weske. 2015. Prediction of business process durations using non-Markovian stochastic Petri nets. *Information Systems* 54 (2015), 1–14.

[20] Arik Senderovich, Matthias Weidlich, Avigdor Gal, and Avishai Mandelbaum. 2014. Queue Mining - Predicting Delays in Service Processes. In *CAiSE*. 42–57.

[21] Niek Tax, Ilya Verenich, Marcello La Rosa, and Marlon Dumas. 2017. Predictive business process monitoring with LSTM neural networks. In *CAiSE*. Springer, To appear.

[22] Wil M P van der Aalst, M Helen Schonenberg, and Minseok Song. 2011. Time prediction based on process mining. *Information Systems* 36, 2 (2011), 450–475.

[23] Sjoerd van der Spoel, Maurice van Keulen, and Chintan Amrit. 2012. Process prediction in noisy data sets: a case study in a dutch hospital. In *International Symposium on Data-Driven Process Discovery and Analysis*. Springer, 60–83.

[24] Boudewijn F van Dongen, Ronald A Crooy, and Wil M P van der Aalst. 2008. Cycle time prediction: when will this case finally be finished?. In *CoopIS*. Springer, 319–336.

[25] Ilya Verenich, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, and Chiara Di Francescomarino. 2016. Minimizing Overprocessing Waste in Business Processes via Predictive Activity Ordering. In *CAiSE*. 186–202.

[26] Yong Yang, Marlon Dumas, Luciano García-Bañuelos, Artem Polyvyanyy, and Liang Zhang. 2012. Generalized aggregate Quality of Service computation for composite services. *Journal of Systems and Software* 85, 8 (2012), 1818–1830.