Received <day> <Month>, <year>; | Revised <day> <Month>, <year>; | Accepted <day> <Month>, <year>

DOI: xxx/xxxx

SPECIAL ISSUE PAPER

Predicting Process Performance: A White-Box Approach

Ilya Verenich*[1,2] | Marlon Dumas[2,1] | Marcello La Rosa[3] | Hoang Nguyen[1] | Arthur ter Hofstede[1]

[1]School of Information Systems, Queensland University of Technology, Brisbane, Australia
[2]Institute of Computer Science, University of Tartu, Tartu, Estonia
[3]School of Computing and Information Systems, University of Melbourne, Victoria, Australia

Correspondence
*Corresponding author name, Corresponding address. Email: ilya.verenich@qut.edu.au

Present Address
Present address

Abstract

Predictive business process monitoring methods exploit historical process execution logs to provide predictions about running instances of a process. These predictions 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 for running instances of a process, including 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 develops this idea in the context of predicting the remaining cycle time of ongoing process instances. The proposed approach has been evaluated on real-life event logs and compared against several baselines.

KEYWORDS:
Process Mining, Predictive Process Monitoring, Flow analysis

1 | INTRODUCTION

Predictive business process monitoring techniques seek to predict the future state or properties of ongoing executions of a process based on models extracted from historical event logs. A wide range of predictive business process monitoring techniques have been proposed to predict for example compliance violations [1,2], the next activity or the remaining sequence of activities of a process instance [3,4], or quantitative process performance indicators, such the remaining cycle time of a process instance [5,6,7]. 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 article 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. For example, the cycle time of a process instance with sequentially performed activities 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 article 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 [8]. 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

a performance indicator for an ongoing process instance, we first apply regression models to estimate the performance of each activity that might potentially be executed within this process instance, and then we 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 article presents the proposed approach and its evaluation in the specific context where the performance indicator to be predicted is the *remaining time* of a process instance. However, the method could be adapted to predict other quantitative performance indicators, provided that the performance indicator can be calculated for each process activity separately.

This article is an extended and revised version of an earlier conference paper [9]. With respect to the conference version, the main additional contribution is a new variant of the proposed remaining time prediction method, namely the *adaptive flow analysis* method. This new method can handle situations where there is not enough data to build a reliable regression model, which happens for example when an activity occurs infrequently in a given state, and hence it has not been observed enough times to build a reliable regression model to predict its execution time. Furthermore, in this article, we have enhanced the experimental setup with additional datasets, additional features for building the regression models, and an alternative machine learning technique, namely XGBoost.

The remainder of the article 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 background concepts and techniques used in the article. 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 article and outlines future work directions.

2 | RELATED WORK

A wide range of predictive business process monitoring problems have been studied in previous work. A recent survey of this field by Márquez-Chamorro et al. [10] identified 39 distinct proposals, out of which 17 focus on the prediction of time-related properties of running process instances (herein called *cases*).

One group of techniques in this field address the problem of predicting whether or not a given case will complete on time, i.e. predicting deadline violations in a boolean manner and with respect to a given *maximum allowed duration* of a case. Pika et al. [11] address this prediction problem by training decision trees over a small number of features, such as which activities have occurred so far and which resources have performed these activities. Conforti et al [12] 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. Metzger et al. [13] tackle the problem of predicting deadline violations by finding correlations between "late show" events and external variables related to weather conditions or road traffic, and then setting thresholds on these external variables (e.g. when the external variable exceeds a threshold, a late show event is predicted).

The problem addressed in this article differs from the one addressed in the above studies because our goal is not to predict whether a given deadline will be fulfilled or violated, but to predict the actual remaining time. In other words, this article addresses a regression problem, whereas the above studies address a classification problem. Building white-box (interpretable) models for classification problems is a different problem than building such models for regression problems. Other related work addressing classification problems in the context of predictive process monitoring include the one by Maggi et al. [1] and by Leontjeva et al. [2], who propose methods 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 at each point in the sequence.

One of the earliest studies addressing the problem of predicting the remaining cycle time of running cases of a business process is the one by Van Dongen et al. [14] In this work, the authors 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. Van der Aalst et al. [5] 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. [15] refine this method by proposing a data-aware transition system wherein each state is annotated with a classifier to predict the next state and each transition is annotated with a regressor to predict the remaining time from that state. Rogge-Solti and Weske [6,7] model business processes as stochastic Petri nets and perform Monte Carlo simulation to predict the remaining time of a case. De Leoni et al. [16] propose a general framework to predict various characteristics of running cases, including the remaining time, based on correlations with other characteristics and using regression trees. Senderovich et al. [17] extend the approach presented by Aalst et al. [5] by incorporating features extracted from queuing networks derived from the event logs. These additional features enhance the accuracy of the regression models in the case where part of the execution time comes from resource contention (i.e. resources being too busy). Finally, Senderovich et al. [18] provide a comprehensive study of how contextual attributes capturing different levels of inter-case dependencies enable more accurate prediction of remaining time.

The methods mentioned above are "black-box" methods, insofar as they predict a single scalar value (the remaining time) without decomposing this prediction into more elementary components. In the case of regression models such as those used in Van Dongen et al. [14], it is possible to extract regression coefficients that tell us to what extent each feature in the model contributes to the prediction (e.g. to what

extent the prediction can be explained by the type of customer, or by the fact that a given activity has been performed multiple times). However, these models do not tell us where the predicted remaining time will be spent. In contrast, the aim of the research reported here is to build white-box prediction models, which enable process workers and managers to understand in which parts of the process the remaining time will be spent. Concretely, in the approach proposed in this article, the remaining cycle time of a case is decomposed into a (weighted) sum of the predicted cycle times of the activities that are yet to be performed in the case, thus allowing process workers to determine to what extent each activity contributes to the prediction.

The problem of predicting remaining time has also been extensively studied in the context of software development processes. For example, Kikas et al.[19] predict issue resolution time in Github projects using static, dynamic and contextual features. A later work by Rees-Jones et al.[20] further emphasizes the importance of contextual features for predicting issue lifetime in Github projects. However, these proposals do not seek to explain in which parts of the process the predicted remaining time will be spent.

Project management software tools incorporate functionality to predict completion dates for software development projects. For example, FogBugz[1] uses Evidence-Based Scheduling based on Monte Carlo simulation to estimate completion dates[21]. The resulting estimates can be traced down to individual tasks in the project. Our work is inspired by these latter approaches but extends them in order to handle business processes with arbitrary structure. One of the main differences in this setting is that business processes have conditional branching and (conditional) loops, while project plans such as those taken as input in FogBugz, do not contain conditional constructs. Also, our approach uses supervised learning techniques in order to be able to take as input as much data as possible (e.g. data attributes and resources involved in the execution of an ongoing case).

A final group of techniques in the field of predictive process monitoring aim to predict future event(s) of a running case. Lakshmanan et al.[22] use Markov chains to estimate the probability of future execution of a given task in a running case; Breuker et al.[23] use probabilistic finite automata to predict the next activity to be performed, while Tax et al.[24] predict the entire continuation of a running case as well as timestamps of future events using long short-term memory (LSTM) neural networks. These related studies, however, do not address the problem of predicting the remaining time. An exception is the technique by Tax et al.[24], which predicts the timestamp of each remaining activity and hence can be seen as a white-box technique to predict the remaining time – since the timestamp of the last activity in the predicted sequence is the predicted end time of the case. However, the experiments reported by Tax et al.[24] show that the proposed method for predicting the remaining sequence has a low accuracy, which hinders its applicability in practice.

[1]http://www.fogcreek.com

3 | BACKGROUND

Predictive process monitoring is a multi-disciplinary area that draws concepts from business process management and process mining on the one side and data mining and machine learning on the other. In this section, we introduce concepts from the aforementioned disciplines that are used in later sections of this paper.

3.1 | Event Logs, Traces and Sequences

Business processes are generally supported by information systems that record data about each individual execution of a process (also called a *case*). Each case consists of a number of *events* representing the execution of activities in a process. Each event has a range of attributes of which three are mandatory, namely (i) *case identifier* specifying which case generated this event, (ii) the *event class* (or *activity name*) indicating which activity the event refers to and (iii) the *timestamp* indicating when the event occurred[2]. An event may carry additional attributes in its payload. For example, in a patient treatment process in a hospital, the name of a responsible nurse may be recorded as an attribute of an event referring to activity "Perform blood test". These attributes are referred to as *event attributes*, as opposed to *case attributes* that belong to the case and are therefore shared by all events relating to that case. For example, in a patient treatment process, the age and gender of a patient can be treated as a case attribute. In other words, case attributes are static, i.e. their values do not change throughout the lifetime of a case, as opposed to attributes in the event payload, which are dynamic as they change from an event to the other.

Formally, an event record is defined as follows:

Definition 1 (Event). An $event$ is a tuple $(a, c, t, (d_1, v_1), \ldots, (d_m, v_m))$ where a is the activity name, c is the case identifier, t is the timestamp and $(d_1, v_1) \ldots, (d_m, v_m)$ (where $m \geq 0$) are event attribute names and the corresponding values assumed by them.

Let $\mathcal{E}$ be the event universe, i.e., the set of all possible event identifiers, and $\mathcal{T}$ the time domain. Then there is a function $\pi_{\mathcal{T}} \in \mathcal{E} \rightarrow \mathcal{T}$ that assigns timestamps to events.

The sequence of events generated by a given case forms a *trace*. Formally,

Definition 2 (Trace). A $trace$ is a non-empty sequence $\sigma = \langle e_1, \ldots, e_n \rangle$ of events such that $\forall i \in [1..n], e_i \in \mathcal{E}$ and $\forall i, j \in [1..n]$ $e_i.c = e_j.c$. In other words, all events in the trace refer to the same case.

A set of *completed traces* (i.e. traces recording the execution of completed cases) comprises an *event log*.

Definition 3 (Event log). An event log L is a set of completed traces, i.e., $L = \{\sigma_i : \sigma_i \in \mathcal{S}, 1 \leq i \leq K\}$, where $\mathcal{S}$ is the universe of all possible traces and K is the number of traces in the event log.

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

As a running example, let us consider an extract of an event log originating from an insurance claims handling process (Table 1). The activity name of the first event in case 1 is *A*, it was completed on *1/1/2017* at *9:13AM*. The additional event attributes show that the cost of the activity was 15 units and the activity was performed by *John*. These two are event attributes. The events in each case also carry two case attributes: the age of the applicant and the channel through which the application has been submitted. The latter attributes have the same value for all events of a case.

Event and case attributes can be categorized into at least two data types – numeric (quantitative) and categorical (qualitative)[25]. With respect to the running example, numeric attributes are *Age* and *Cost*, while categorical attributes are *Channel*, *Activity* and *Resource*. Each data type requires different preprocessing to be used in a predictive model. Specifically, numeric attributes are typically represented as such, while for categorical attributes, one-hot encoding is typically applied.

To illustrate how one-hot encoding works, let us consider a categorical attribute A with $|A|$ possible values, or levels. We take an arbitrary but consistent ordering over the set of levels of A, and use $index \in A \rightarrow \{1, \ldots, |A|\}$ to indicate the position of a given attribute value a in it. The one-hot encoding assigns the value 1 to feature number $index(a)$ and a value of 0 to the other features[24].

As we aim to make predictions for traces of incomplete cases, rather than for traces of completed cases, we define a function that returns the first k events of a trace of a (completed) case.

Definition 4 (Prefix function). Given a trace $\sigma = \langle e_1, \ldots, e_n \rangle$ and a positive integer $k \leq n$, $hd^k(\sigma) = \langle e_1, \ldots, e_k \rangle$.

For example, for a sequence $\sigma_1 = \langle a, b, c, d, e \rangle$, $hd^2(\sigma_1) = \langle a, b \rangle$.

The application of a prefix function will result in a *prefix log*, where each possible prefix of an event log becomes a trace.

Definition 5 (Prefix log). Given an event log L, its prefix log L^* is the event log that contains all prefixes of L, i.e., $L^* = \{hd^k(\sigma) : \sigma \in L, 1 \leq k \leq |\sigma|\}$.

For example, a complete trace consisting of three events would correspond to three traces in the prefix log – the partial trace after executing the first, the second and the third event.

3.2 | Process Models

Process models provide a visual representation of the underlying business process. A process model consists of a set of activities and their structuring using directed control flow edges and gateway nodes that implement process routing decisions[26]. Formally,

Definition 6 (Process model). A process model is a tuple $(N, E, type)$, where $N = N_A \cup N_G$ is a set of nodes (N_A is a nonempty set of activities and N_G is a set of gateways; the sets are disjoint), $E \subseteq N \times N$ is a set of directed edges between nodes defining control flow, $type$ is a function $N_G \rightarrow \{AND, XOR, OR\}$ is a function that assigns a control flow construct to each gateway.

Many modeling notations have been proposed to represent process models, including Event-driven Process Chains (EPC), UML Activity Diagrams and Petri nets. Nevertheless, the definition above is rather generic and abstracts from the specific notation.

One of the most common process modeling notations is the Business Process Model and Notation (BPMN) developed by the Object Management Group (OMG). Figure 1 shows a subset of the core elements of BPMN. The *start* and *end* events represent the initiation and termination of an instance of a process respectively. The tasks denote activities to be performed. The flow represents the order among the events, gateways and tasks. Finally, *gateways* are control flow elements and they can represent either the splitting or merging of paths. In the case of *exclusive gateways* (also known as XOR-gateways), a split has more than one outgoing flow, but only one of them can be activated, according to a pre-defined condition. Its counterpart, the join exclusive gateway, merges the incoming alternative flows. Conversely, a *parallel gateway* (also known as AND-gateways) denotes the parallel activation of all the outgoing branches; whereas, the merging counterpart denotes the synchronization of the multiple incoming paths[27].

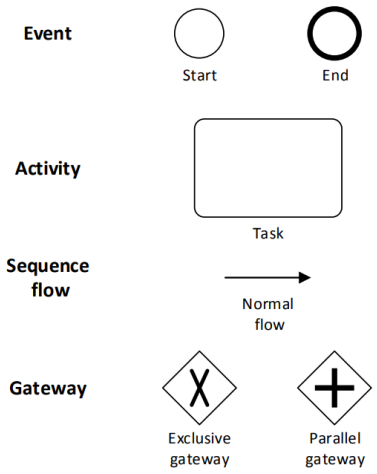

FIGURE 1 Core BPMN elements[27].

A process model can be decomposed into process fragments. A process fragment is a connected part of a process model. For the purpose of this paper, we require a fragment to have a single entry and a single exit point. Such fragments are referred to as *SESE fragments*. From a modeling perspective, SESE fragments are very handy: structurally every SESE fragment can be replaced with one aggregating activity. A model that can be decomposed into SESE fragments, without impacting the behavior it describes, is called a block-structured, or a *structured* model.

A process model can be provided by the process stakeholders or can be automatically discovered from the corresponding event log via a process discovery algorithm[28].

Definition 7 (Process discovery algorithm). Let L be an event log as specified in Definition 3. A process discovery algorithm is a function that maps L onto a process model such that the model is "representative" for the behavior seen in the event log.

TABLE 1 Extract of an event log.

Case	Case attributes		Event attributes			
ID	Channel	Age	Activity	Timestamp	Resource	Cost
1	Email	37	A	1/1/2017 9:13:00	John	15
1	Email	37	B	1/1/2017 9:14:20	Mark	25
1	Email	37	D	1/1/2017 9:16:00	Mary	10
1	Email	37	C	1/1/2017 9:18:00	Mark	10
1	Email	37	F	1/1/2017 9:18:05	Kate	20
1	Email	37	G	1/1/2017 9:18:50	John	20
1	Email	37	H	1/1/2017 9:19:00	Kate	15
2	Email	52	A	2/1/2017 16:55:00	John	25
2	Email	52	D	2/1/2017 17:00:00	Mary	25
2	Email	52	B	3/1/2017 9:00:00	Mark	10
2	Email	52	C	3/1/2017 9:01:00	Mark	10
2	Email	52	F	3/1/2017 9:01:50	Kate	15

A wide range of process automated process discovery algorithms have been proposed in the literature [29], with Petri nets and BPMN being the most common output model formats.

3.3 | Flow Analysis

Flow analysis is a family of techniques that enables estimation of the overall performance of a process given knowledge about the 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 [8]. Since flow analysis is typically applied to structured process models described in the BPMN notation, the estimation can be easily explained in terms of its elementary components.

Definition 8 (Cycle time of an activity). A *cycle time* of an activity i is the 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 on 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 may choose to batch all applications and check them only once at the start or the end of a working day [8].

To understand how flow analysis works, we start with an example of a process with *sequential* SESE fragments of events as in Figure 2 a. Each fragment 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 fragments is the sum of the cycle times of each fragment [8]:

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

Let us now consider a process with a decision point between N mutually exclusive fragments, represented by an *XOR gateway* (Figure 2 b). In this case, the average cycle time of the process is determined by:

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

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 gateways where activities can be executed concurrently as in Figure 2 c, 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 2 d. The fragment is executed once. Next, it may be repeated each time with a probability r referred to as the *rework probability*. The number of times

that the rework fragment will be executed follows a geometric distribution with the expected value $1/(1-r)$. Thus, the average cycle time of the fragment in this case is:

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

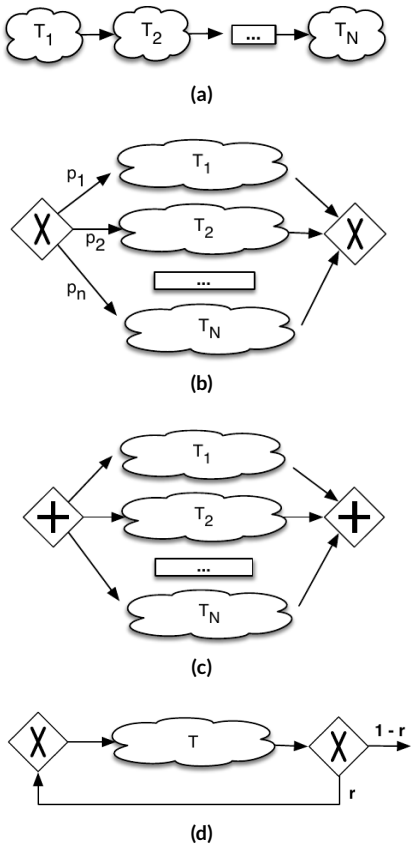

FIGURE 2 Typical process model patterns: sequential (a), XOR-block (b), AND-block (c) and rework loop (d).

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 2 c 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-gateway is traversed, every branch in the AND join is executed and therefore the costs of these branches add up to one another [8].

In case of structured process models, 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 [30].

Importantly, flow analysis equations were defined when the *average* cycle time of the process is in question. However, the same equations can also be used to predict the *remaining* cycle time of a given ongoing instance σ based on the information available in the prefix $hd^k(\sigma)$. For example, in Equation 2, branching probabilities p_i returned by the predictor can serve as confidence values [31]. Thus, among the predicted cycle times T_i for each fragment, the highest weight is given to the one that corresponds to the most likely continuation of σ.

3.4 | Supervised Machine Learning

Machine learning is a research area of computer science concerned with the discovery of models, patterns, and other regularities in data [32]. Closely related to machine learning is data mining. Data mining is the "core stage of the *knowledge discovery process* that is aimed at the extraction of interesting – non-trivial, implicit, previously unknown and potentially useful – information from data in large databases" [33]. Data mining techniques focus more on exploratory data analysis, i.e. discovering unknown properties in the data, and are often used as a preprocessing step in machine learning to improve model accuracy.

3.4.1 | Overview

A machine learning system is characterized by a learning algorithm and training data. The algorithm defines a process of learning from information extracted, usually as *features vectors*, from the training data. In this work, we will deal with *supervised* learning, meaning training data is represented as n labeled samples:

$$D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) : n \in \mathbb{N}\}, \tag{5}$$

where $\mathbf{x}_i \in \mathcal{X}$ are m-dimensional feature vectors ($m \in \mathbb{N}$) and $y_i \in \mathcal{Y}$ are the corresponding labels, i.e. values of the target variable.

Feature vectors extracted from the labeled training data are used to fit a predictive model that assigns labels to new data given labeled training data while minimizing error and model complexity. In other words, a model generalizes the pattern, providing a mapping $\mathcal{X} \to \mathcal{Y}$. The labels can be either continuous, e.g. cycle time of activity, or discrete, e.g. loan grade. In the former case, the model is referred to as regression; while in the latter case we are talking about a classification model.

From a probabilistic perspective, the machine learning objective is to infer a conditional distribution $P(\mathcal{Y}|\mathcal{X})$. A standard approach to tackling this problem is to represent the conditional distribution with a parametric model, and then to obtain the parameters using a training set containing $\{\mathbf{x}_n, y_n\}$ pairs of input feature vectors with corresponding target output vectors. The resulting conditional distribution can be used to make predictions of y for new values of $\mathbf{x}$.

3.4.2 | XGBoost

As a learning algorithm, in this paper, we apply the so-called eXtreme Gradient Boosting (XGBoost) [34], one of the latest implementations of Gradient Boosting Machines (GBM), which has been successfully utilized across various domains [35,36,37] as well as in machine learning competitions such as Kaggle[3]. Olson et al. [38] performed a thorough analysis of 13 state-of-the-art, commonly used machine learning algorithms on a set of 165 publicly available classification problems and found that gradient tree boosting achieves the best accuracy for the highest number of problems.

XGBoost is based on the theory of boosting, wherein the predictions of several "weak" learners (models whose predictions are slightly better than random guessing), are combined to produce a "strong" learner [39]. In GBMs, these "weak" learners are combined by following a gradient learning strategy. At the beginning of the calibration process, a "weak" learner is fit to the whole space of data, and then, a second learner is fit to the residuals of the first one. This process of fitting a model to the residuals of the previous one goes on until some stopping criterion is reached. Finally, the output of the GBM is a kind of weighted mean of the individual predictions of each weak learner. Regression trees are typically selected as "weak" learners [40].

The main challenge of boosting learning in general, and of GBMs in particular, is the risk of overfitting, as a consequence of the additive calibration process. Typically, this issue is faced by including different regularization criteria in order to control the complexity of the algorithm. However, the computational complexity of these type of mechanisms is rather high [40]. In this context, the main objective of XGBoost is to control the complexity of the algorithm without excessively increasing the computational cost. To this end, XGBoost aims to minimize the following *Loss+Penalty* objective function:

$$R(\Theta) = \sum_{i=1}^{n} l(y_i, \hat{y_i}) + \sum_{i=1}^{k} \Omega(f_i), \tag{6}$$

where Θ includes the model parameters to be learned during training, l is the loss function, which measures the cost between ground truth y_i and prediction $\hat{y_i}$, and $\Omega(f_i)$ is the regularization term. The loss function for the predictive term can be specified by the user, and it is always truncated up to the second term of its Taylor series expansion for computational reasons. The regularization term is obtained with an analytic expression based on the number of leaves of the tree and the scores of each leaf. The key point of the calibration process of XGBoost is that both terms are ultimately rearranged in the following expression [40]:

$$R(\Theta) = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T, \tag{7}$$

where G and H are obtained from the Taylor series expansion of the loss function, λ is the L2 regularization parameter and T is the number of leaves. This analytic expression of the objective function allows a rapid scan from left to right of the potential divisions of the tree, but always taking into account the complexity.

[3]https://www.kaggle.com

3.4.3 | Hyperparameter optimization

Each prediction technique is characterized by model parameters and by hyperparameters [41]. While model parameters are learned during the training phase so as to fit the data, hyperparameters are set outside the training procedure and are used for controlling how flexible the model is in fitting the data. For instance, the number of weak learners is a hyperparameter of the XGBoost algorithm. The impact of hyperparameter values on the accuracy of the predictions can be extremely high. Optimizing their value is therefore important, but, at the same time, optimal values depend on the specific dataset under examination [41].

A traditional way of performing hyperparameter optimization is *grid search*, which is an exhaustive search through a manually specified subset of the hyperparameter space of a learning algorithm. A grid search algorithm must be guided by some performance metric, typically measured by cross-validation on the training set or evaluation on a held-out validation set [32]. However, in a large, multi-dimensional, search space, *random search* is often more effective, given a fixed number of iterations. [41] In random search, hyperparameters are sampled from a random uniform distribution.

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 3).

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 3.3. 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

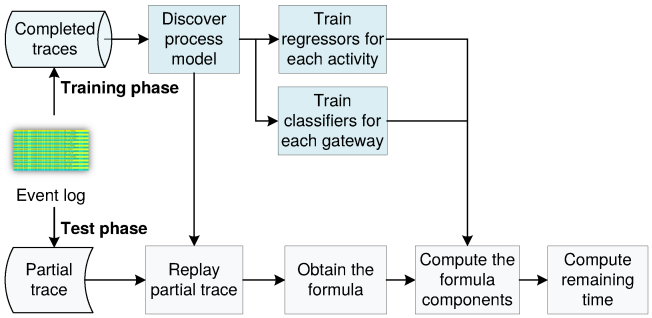

FIGURE 3 Overview of the proposed approach.

process, generally we need to discover the model from event logs. For that, we use a two-step automated process discovery technique proposed in[42] 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 [4] as well as a ProM plugin, namely *StructuredMiner*.

The technique in[42] pursues a two-phase "discover and structure" approach. In the first phase, a model is discovered from the log using either the Heuristics Miner or Fodina Miner. They have 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.

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[43] which is available as a plugin for the open-source process analytics platform Apromore[5].

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, 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[43], 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 4). 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, based on costs accumulated along the path from the root to the current node. If the alignment is good, 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)
```

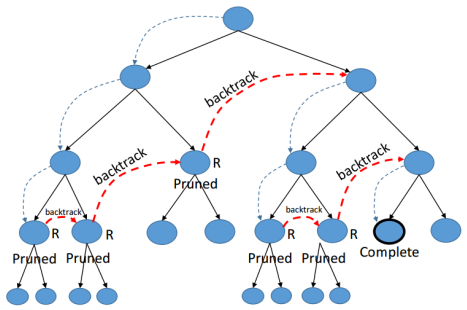

FIGURE 4 Backtracking algorithm (taken from[43]).

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 5 . Applying the flow analysis formulas described in Section 3.3, the *average* cycle time of this process can be decomposed as follows:

$$CT = T_A + \max(T_B + T_C, T_D) + T_F + p2\left(T_G + \frac{T_H}{1-r}\right) \quad (8)$$

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

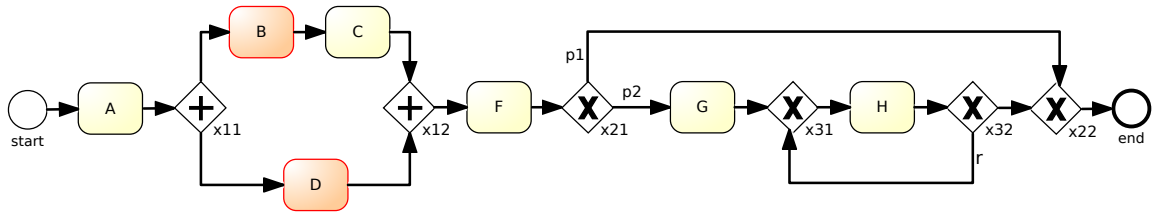

FIGURE 5 Example process model. Highlighted is the current marking

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, for the remaining cycle time of $hd(\sigma)$, we obtain the formula:

$$\begin{aligned} CT_{rem} &= 0 + \max(0 + T_C, 0) + T_F + p2 \left(T_G + \frac{T_H}{1-r} \right) = \\ &= T_C + T_F + p2 \left(T_G + \frac{T_H}{1-r} \right). \end{aligned} \tag{9}$$

Since activities A, B and D have already been executed, they do not contribute to the *remaining* cycle time. Thus, they are not a part of the formula. All the other formula terms need to be predicted using the data from $hd(\sigma)$.

Similarly, if a current marking is inside an 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 Figure 6 . 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 5), 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 an 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:

$$CT_H = \mathbf{T_{H'}} + r \frac{T_H}{1-r}, \tag{10}$$

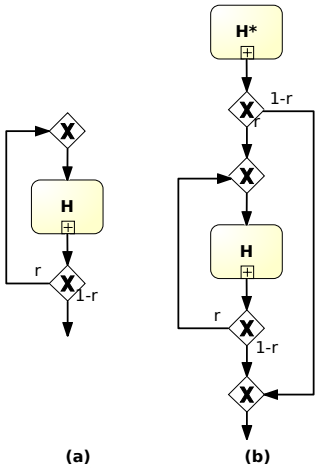

FIGURE 6 Unfolding the rework loop of F

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, we can obtain these estimates 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*

3. By combining the above two approaches as follows: If the number of training cases to fit the prediction model for a given

activity A is less than a certain threshold N_0, then we find the mean cycle times and branching probabilities as in the mean flow analysis method. Otherwise, we fit the predictors as in predictive flow analysis. This hybrid approach is herein called *adaptive flow analysis*. The value N_0 can be treated as a hyperparameter and is to be determined during the hyperparameter optimization procedure.

The rationale for the adaptive flow analysis is that in a high dimensional feature space, the number of observations to train a model may not be enough, thus the predictions may be unreliable and unstable [44]. This may happen when we predict cycle times of rare activities or branching probabilities of gateways that are rarely taken. Furthermore, 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. In such cases, we can then use the mean historical activity cycle times and mean probabilities instead.

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 insufficient data available to base the predictions upon.

4.6 | Feature encoding

In order to fit predictive models, we need to encode process execution traces in the form of fixed-length feature vectors. In the related work, there have been proposed two common approaches to achieve this:

1. Training a single predictive model using case attributes as well as aggregated attributes of executed events. This approach has been used in [1,14,16,24], among others. Typical aggregation functions applied over event attributes include *count* (for categorical attributes) and *mean* (for numeric ones).

2. Training multiple predictive models, one for each possible prefix length. In this way, values of each event attribute need not be aggregated across the partial case, therefore, they may be preserved, as well as the order of attributes. In other words, this encoding is lossless as compared to the previous one. This approach has been used in [2,31,9], among others.

Although lossless, the latter approach has been shown to be outperformed by a single predictive model in some circumstances [45]. This is due to the fact that combining the knowledge from all events performed so far may provide more signal than using the order of events and their individual attributes. In order to quantify this phenomenon in the context of flow analysis, in our work we consider both approaches.

4.6.1 | Feature encoding with a single predictive model

In a *single predictive model* approach, we create feature vectors by extracting the following data from each prefix of the corresponding prefix log:

- Case attributes. These attributes are static in the sense that they do not change as the case progresses. As such, they can simply be appended to feature vectors.

- Aggregated event attributes. As event attributes are dynamic, i.e. each event in a trace has its own attribute values, in order to encode them in a fixed-length vector, we apply several aggregation functions. For numeric attributes, we compute their mean, minimum and maximum values across a partial case, as well as their sum and standard deviation. For categorical attributes, we count how many times a specific level has appeared (e.g. how many times a specific activity has been executed, or how many activities a specific resource has performed).

To fit a predictive model, we append to these feature vectors the value of the target variable y that is to be learned. For example, if we are to predict the cycle time of an activity, we calculate it as the time difference (in seconds) between the completion timestamp of that activity and the completion timestamp of the previous activity. If the activity is never executed in a given case, 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.

As a running example, we consider a snapshot of the log with two complete cases in Table 1 that corresponds to the process model in Figure 5 . The events are ordered according to their completion timestamp. Table 2 illustrates the resulting training set that will be composed for the prediction of a cycle time of activity F. For convenience, we replaced absolute timestamps with relative ones (in seconds), starting from the beginning of a case. Furthermore, for simplicity, we showed only one aggregation function, sum, for numerical attributes.

Similarly, to predict branching probabilities, we assign a class label to each outgoing branch. For example, if we are to predict the branching probabilities for the $X32$ gateway, we can assign class 0 to the branch that leads to rework and class 1 to the other branch. Evidently, the probability of class 0 would be equal to the rework probability r. Thus, for the first case in Table 1 $X32 = 1$, while for the second case $X32$ is undefined since the case does not reach that decision point. Consequently, the second case cannot be used to populate a training set for the prediction of $X32$. Table 3 shows the resulting training set.

From these examples, it is evident that the dimensionality of feature vectors is constant for a given log and depends on: (i) the number of case attributes, (ii) the number of categorical event attributes and the number of possible values, or levels, of each such attribute, (iii) the number of numeric event attributes and the number of aggregation functions applied for them.

TABLE 2 Training set to predict the cycle time of F (single model).

Channel	Age	Activity_A	Activity_B	Activity_C	Activity_D	Activity_F	Activity_G	Activity_H	Resource_John	Resource_Mark	Resource_Mary	Resource_Kate	sum_Timestamp	sum_Cost	. . .	Target
Email	37	1	0	0	0	0	0	0	1	0	0	0	0	15		5
Email	37	1	1	0	0	0	0	0	1	1	0	0	80	40		5
Email	37	1	1	0	1	0	0	0	1	1	1	0	180	50		5
Email	37	1	1	1	1	0	0	0	1	2	1	0	300	60		5
Email	37	1	1	1	1	1	0	0	1	2	1	1	305	80		5
Email	37	1	1	1	1	1	1	0	2	2	1	1	350	100		5
Email	37	1	1	1	1	1	1	1	2	2	1	2	360	115		5
Email	52	1	0	0	0	0	0	0	1	0	0	0	0	25		50
Email	52	1	0	0	1	0	0	0	1	0	1	0	300	50		50
Email	52	1	1	0	1	0	0	0	1	1	1	0	57900	60		50
Email	52	1	1	1	1	0	0	0	1	2	1	0	57960	70		50
Email	52	1	1	1	1	1	0	0	1	2	1	1	58010	85		50

TABLE 3 Training set to predict the branching probabilities of $X32$ (single model).

Channel	Age	Activity_A	Activity_B	Activity_C	Activity_D	Activity_F	Activity_G	Activity_H	Resource_John	Resource_Mark	Resource_Mary	Resource_Kate	sum_Timestamp	sum_Cost	. . .	Target
Email	37	1	0	0	0	0	0	0	1	0	0	0	0	15		1
Email	37	1	1	0	0	0	0	0	1	1	0	0	80	40		1
Email	37	1	1	0	1	0	0	0	1	1	1	0	180	50		1
Email	37	1	1	1	1	0	0	0	1	2	1	0	300	60		1
Email	37	1	1	1	1	1	0	0	1	2	1	1	305	80		1
Email	37	1	1	1	1	1	1	0	2	2	1	1	350	100		1
Email	37	1	1	1	1	1	1	1	2	2	1	2	360	115		1

4.6.2 | Feature encoding with multiple predictive models

In a *multiple predictive model* approach, we concatenate 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. In general, for a case with U case attributes $\{s_1, \ldots, s_U\}$ containing k events $\{e_1, \ldots, e_k\}$, each of them having an associated payload $\{d_1^1, \ldots, d_1^R\}, \ldots \{d_k^1, \ldots, d_k^R\}$ of length R, the resulting feature vector would be:

$$\vec{X} = (s_1, \ldots, s_U, e_1, d_1^1, \ldots, d_1^R, \ldots, e_k, d_k^1, \ldots, d_k^R) \quad (11)$$

With this encoding, the length of the feature vector increases with each executed event e_k:

$$U + k \cdot R \quad (12)$$

Consequently, this approach requires fitting a separate model for each possible length of a test prefix. Table 4 provides an example of the resulting training set for the prediction of the cycle time of F for prefixes of length $k = 3$.

Analogously, Table 5 provides an example of the resulting training set for the prediction of the branching probabilities $X32$ for prefixes of length $k = 3$. The set includes only one case for which the decision point has been actually taken.

As compared to the single model approach where each sample of a prefix log created from the training set becomes a training sample (Table 2), in the multiple model approach, each training trace produces only one sample. It should be noted that if a case does not reach length k, i.e. it finishes earlier, there are two options to proceed:

- Discard such cases from the training set for prefix lengths k
- Impute missing event attribute values with zeros or their historical averages computed from cases that have at least k events, so that the resulting feature vectors' dimensionality would be determined by Eq. 12. This approach is often referred to as *padding* in machine learning[46].

In this work, we will use the former approach, as we are mostly interested in predictions in the *early* stages of a process evolution where many process instances have not finished yet, so there is still sufficient amount of data to train the predictive models.

5 | EVALUATION

In the following section, we empirically compare the proposed flow analysis-based approaches with each other and with various baselines proposed in previous work. In particular, we seek to answer the following research questions:

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

RQ2. Does the adaptive flow analysis approach provide added value over the mean and predictive flow analysis approaches?

The source code and supplementary material required to reproduce the experiments discussed in the rest of this section can be found at http://github.com/verenich/flow-analysis-predictions.

5.1 | Datasets

We conducted the experiments using nine real-life event datasets. To ensure the reproducibility of the experiments, the logs we used are publicly available at the "4TU Center for Research Data"[6] as of September 2017, except for one log, which we obtained from the demonstration version of a software tool. Table 6 summarizes the basic characteristics of each dataset.

[6] https://data.4tu.nl/repository/collection:event_logs_real

TABLE 4 Training set to predict the cycle time of $F, k = 3$ (multiple models).

Channel	Age	Activity_1	Timestamp_1	Resource_1	Cost_1	Activity_2	Timestamp_2	Resource_2	Cost_2	Activity_3	Timestamp_3	Resource_3	Cost_3	Target
Email	37	A	0	John	15	B	80	Mark	25	D	180	Mary	10	5
Email	52	A	0	John	25	D	300	Mary	25	B	57900	Mark	10	50

TABLE 5 Training set to predict the branching probabilities of $X32, k = 3$ (multiple models).

Channel	Age	Activity_1	Timestamp_1	Resource_1	Cost_1	Activity_2	Timestamp_2	Resource_2	Cost_2	Activity_3	Timestamp_3	Resource_3	Cost_3	Target
Email	37	A	0	John	15	B	80	Mark	25	D	180	Mary	10	1

BPIC'12. This event log originates from the Business Process Intelligence Challenge [7] held in 2012 and contains data from the application procedure for financial products at a large financial institution. The 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 that tracks the states of work items associated with the application (BPIC'12 W). For the purpose of this work, we treat these three subprocesses as separate datasets. Additionally, the BPIC'12 W subprocess 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 3.3, flow analysis approaches 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, the error will propagate. To verify our hypothesis, we modify the log by 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. In the rest of the paper, we refer to the modified log as BPIC'12 W_n|1|.

[7] doi:10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f

Credit Requirement (CR). This log contains information about a credit requirement process in a bank.[8] It contains data about events, time execution, etc. A distinctive feature is that all cases follow the same path, thus the process model is sequential and does not contain gateways.

Helpdesk. This log contains events from a ticketing management process of the help desk of an Italian software company.[9] 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.

Hospital. This log was obtained from the financial modules of the ERP system of a regional hospital.[10] The log contains events that are related to the billing of medical services that have been provided by the hospital. Each trace of the event log records the activities executed to bill a package of medical services that were bundled together. The log is a random sample of process instances that were recorded over three years.

Invoice. This log refers to an invoice approval process and comes as a demonstration log with the Minit process intelligence software.[11]

[8] doi:10.4121/uuid:453e8ad1-4df0-4511-a916-93f46a37a1b5
[9] doi:10.17632/39bp3vv62t.1
[10] doi:10.4121/uuid:76c46b83-c930-4798-a1c9-4be94dfeb741
[11] https://www.minit.io

TABLE 6 Statistics of the datasets used in the experiments.

Dataset	Number of					Mean	Mean case
	cases	activities	case variants	case attributes	event attributes	events per case	duration (days)
BPIC'12 A	12,007	10	10	1	4	4.49	7.5
BPIC'12 O	3,487	7	6	1	4	4.56	15.1
BPIC'12 W	9,650	6	2,263	1	5	7.50	11.4
BPIC'12 W_n|1|	9,650	6	71	1	5	3.05	11.4
CR	10,036	8	1	0	7	8	0.95
Helpdesk	3,218	5	8	7	4	3.30	7.3
Hospital	59,228	8	5	1	20	5.59	165.2
Invoice	5,233	20	14	5	10	12.27	2.225
RTFMP	81,286	10	10	4	10	5.06	582

RTFMP. The last log describes a road traffic fines management process[12] in an Italian police unit. It contains events related to fine notifications, as well as partial repayments.

All the logs have been preprocessed to remove incomplete cases as well as cases that have not been recorded from their start. Furthermore, as mentioned in Section 3.3, flow analysis cannot readily deal with unstructured models. In addition, the discovery technique described in Section 4.2 aims to mine maximally structured models, meaning that when the business process is inherently unstructured, the resulting model will not be fully structured. Specifically, the technique 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 will decrease the accuracy of the discovered model, which may in turn negatively affect the accuracy of the flow analysis-based predictions of the remaining time. Hence, instead, we remove from the log those cases that cause overlapping loops. Finally, we remove traces that are shorter than 2 events, as we make predictions starting from the second event in the log.

5.2 | Experimental Setup

In this subsection, we describe our approach to split the event logs into training and test sets along the temporal dimension. Next, we provide a description of our evaluation criteria and the baselines to compare against. Finally, we explain the employed hyperparameter optimization procedure.

5.2.1 | Data split

In our experiments, we order the cases in the logs based on the time when the first event of each case has occurred. Then, we split the logs into two parts. We use the first part (80% of each case) as a training set, i.e. as historical data to train the predictive models. The remaining 20% of each case is used to evaluate the accuracy of the predictions. Furthermore, to perform hyperparameter optimization, we create a hold-out validation set from the last 20% of the original training set.

5.2.2 | Evaluation metrics

Two measures commonly employed to assess a predictive process monitoring technique are accuracy and earliness[2,31]. Indeed, in order to be useful, a prediction should be accurate and should be made early on to allow enough time to act upon.

Accuracy

To assess the accuracy 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)[47], where MAE is defined as the arithmetic mean of the prediction errors, RMSE as the square root of the average 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 of the same process, sometimes with values on 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 would be skewed in such situations. Hence, we use MAE to measure the error in predicting the remaining time. Formally,

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y_i}| \tag{13}$$

where $y_i \in \mathcal{Y} = \mathbb{R}$ is an actual value of a function in a given point and $\hat{y_i} \in \mathcal{Y} = \mathbb{R}$ is a predicted value.

Earliness

A common approach to measure the earliness of the predictions is to evaluate the accuracy of the models after each arrived event or at fixed time intervals. Naturally, uncertainty decreases as a case progresses towards its completion. Thus, the earlier we reach the desired level of accuracy, the better the technique is in terms of its earliness.

To this end, we make predictions for prefixes $hd^k(\sigma)$ of traces σ in the test set starting from $k = 2$. However, using all possible values of k is fraught with several issues. Firstly, a large number of prefixes as compared to the number of traces considerably increases the training time of the prediction models. Secondly, for a single model approach, the longer traces tend to produce much more prefixes than shorter ones and, therefore, the prediction model is biased towards the longer cases[45]. Finally, for a multiple model approach, if the distribution of case lengths has a long tail, for very long prefixes, there are not enough traces with that length, and the error measurements become unreliable. Consequently, we use prefixes of up to 10 events only in both the training and the test phase. If a trace contains less than 10 events, we use all prefixes, except the last one, as predictions do not make sense when the case has just completed. In other words, the input for our experiments is a filtered prefix log $L^* = \{hd^k(\sigma) : \sigma \in L, 1 \leq k \leq min(|\sigma| - 1, 10)\}$.

5.2.3 | Baselines

We compare our approach against several baseline approaches. Firstly, we use a black-box approach that predicts the remaining time via regression as a single scalar value using the same feature set and the same learning algorithm as our flow analysis-based approaches. Similarly, we experiment both with the single predictive model and with the multiple predictive model variants, as specified in Section 4.6. The black-box approach with a single model roughly corresponds to the method proposed by de Leoni et al.[16] with decision trees being replaced by XGBoost, so that the results are comparable to our approach (same underlying machine learning technique). Similarly, the black-box approach with multiple models corresponds to the one proposed by Leontjeva et al.[2], with classification being replaced by regression.

[12] doi:10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5

Secondly, we use a transition system (TS) based method proposed by van der Aalst et al.[5] applying both set, bag and sequence abstractions.

Finally, we use the stochastic Petri-net based approach proposed by Rogge-Solti and Weske[6,7]. 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.

5.2.4 | Hyperparameter optimization

In this work, we use the implementation of XGBoost from the `scikit-learn` library[48] for `Python` which allows for a wide range of learning parameters as described in the work by Tianqi and Carlos[34]. To achieve the best predictive power, these parameters need to be tuned for each method and dataset.

To this end, we perform random search to find the best set of hyperparameters for predictive flow analysis. Table 7 lists the most sensitive learning parameters that were optimized during the random search procedure. We perform a total of 50 iterations in the specified search space. The procedure is repeated for each log separately. All other parameters are left to their defaults. In the first stage, for each combination of parameter values, we train a model based on the 80% of the original training set and evaluate its performance on the validation set. In the second stage, we select one combination of the parameters that yields the best performance and retrain a model with these parameters, now using the whole training set.

For the adaptive approach, we treat the number of training samples N as an additional hyperparameter. The relationship between the size of the training set and the dimensionality of the problem has been studied extensively in the machine learning literature[49,44]. Common rules of thumb are that the number of training samples should be $50 + 8m$, $10m$ and m^2, where m is the dimensionality.[50] Accordingly, we define a set of thresholds $N_0 \in \{5m, 10m, 20m, m^2\}$. If $N > N_0$, then we fit a predictive model for a given activity or a gateway, otherwise, we use average historical values of cycle times and branching probabilities respectively in the flow analysis formulas. Similarly to the XGBoost hyperparameters, we choose the value of N_0 that minimizes the validation error for a given log.

To avoid a potential bias towards our approaches, we perform the same procedure for the other two regression-based black-box baselines. The transition system baseline[5] does not have parameters other than the type of abstraction (set, bag of sequence) which we treat as three separate methods. For the stochastic Petri net, we vary the kind of transition distribution and memory semantics. We select the parameters that yield the best performance on the validation set and use them for the test set.

5.3 | Evaluation Results

Table 8 reports the average prediction accuracy 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 can see that in 5 out of 9 datasets, the white-box family of approaches achieve the best results. They are followed by the transition system approaches, which provide the overall best predictions in 2 datasets. Stochastic Petri net and black-box approaches achieve the best result only in 1 dataset each.

Figure 7 presents the prediction accuracy in terms of MAE, evaluated over different prefix lengths. To keep the plots readable, only the transition system-based method out of the three variants (set, bag or sequence) that achieved the highest score according to Table 8 is shown in this Figure. Each evaluation point includes prefix traces of exactly the given length. In other words, traces that are shorter than the required prefix are left out of the calculation. Therefore, the number of cases used for evaluation is monotonically decreasing when increasing the prefix length. In 5 out of 9 datasets, the flow analysis approaches (FA) generally achieve higher accuracy at a fixed stage in the execution of a case. On the other hand, they offer predictions with a required level of accuracy earlier on.

In most of the datasets, we see that the MAE decreases as cases progress. It is natural that the prediction task becomes trivial when cases are close to completion. However, for the *BPIC'12 A* dataset, the predictions become less accurate as the prefix length increases from 2 to 4. This phenomenon is caused by the fact that this dataset contains some short traces for which it appears to be easy to predict the outcome. These short traces are not included in the later evaluation points, as they have already finished by that time. Therefore, we are left with longer traces only, which appear to be more challenging for the classifier, hence decreasing the total accuracy on larger prefix lengths.

As a simple bulk measure to compare the performance of the proposed techniques and the baselines, we plot their mean rankings across all datasets in Figure 8 . Ties were resolved by assigning every tied element to the lowest rank. For example, in *BPIC'12 A* both predictive FA based on a single predictive model and adaptive FA were ranked first, while the next best technique, predictive FA based on multiple models was ranked third. The rankings show the proposed flow analysis approaches, particularly the adaptive variant, consistently outperform the baselines in terms of accuracy (measured by MAE), in addition to providing white-box predictions.

To complement the above observations, we also compare aggregated error values. These values need to be normalized, e.g. by the mean case duration, so that they are on a comparable scale. In order to do that, for each log, we divide average MAE values and their standard deviation across all prefixes reported in Table 8 by the mean case duration for that log reported in Table 6 . The results for each technique are given in Figures 9 and 10 respectively. We can see that adaptive flow analysis has an average error of 48% of the mean case duration across all datasets. In contrast, MAE values for the adaptive flow analysis are less stable across the prefix lengths, as indicated by Figure 10 . In particular,

TABLE 7 Learning parameters of XGBoost that have been tuned.

Parameter	Explanation	Search space
n_estimators	Number of decision trees ("weak" learners) in the ensemble	$[40, 1000]$
learning_rate	Shrinks the contribution of each successive decision tree in the ensemble	$[0.01, 0.07]$
subsample	Fraction of observations to be randomly sampled for each tree.	$[0.5, 1]$
colsample_bytree	Fraction of columns (features) to be randomly sampled for each tree.	$[0.4, 1]$
max_depth	Maximum tree depth for base learners	$[3, 9]$
min_child_weight	Minimum sum of weights of all observations required in a child	$[1, 3]$

TABLE 8 Weighted average MAE over all prefixes.

	BPIC'12 A	**BPIC'12 O**	**BPIC'12 W**	**BPIC'12 W_n|1|**	**CR**
predictive FA (single)	$\mathbf{6.677 \pm 3.72}$	5.95 ± 2.832	6.946 ± 1.057	5.162 ± 1.474	$\mathbf{0.075 \pm 0.039}$
predictive FA (multiple)	6.838 ± 4.155	6.008 ± 2.643	6.823 ± 0.957	5.191 ± 1.472	0.087 ± 0.043
mean FA	7.62 ± 3.528	6.243 ± 3.237	$\mathbf{6.197 \pm 0.54}$	$\mathbf{5.071 \pm 1.428}$	0.339 ± 0.187
adaptive FA	$\mathbf{6.677 \pm 3.72}$	5.95 ± 2.832	6.921 ± 1.057	$\mathbf{5.071 \pm 1.428}$	0.078 ± 0.035
black box (single)	7.325 ± 4.205	5.755 ± 2.727	6.768 ± 0.6	5.353 ± 1.115	0.087 ± 0.043
black box (multiple)	7.329 ± 4.564	5.806 ± 2.713	6.801 ± 0.541	5.563 ± 1.117	0.088 ± 0.847
TS (set)	8.263 ± 5.237	6.517 ± 3.136	7.512 ± 0.628	5.935 ± 1.38	0.358 ± 0.201
TS (bag)	8.263 ± 5.237	6.517 ± 3.136	7.645 ± 0.552	5.932 ± 1.411	0.358 ± 0.201
TS (sequence)	8.263 ± 5.237	6.517 ± 3.136	7.618 ± 0.583	5.937 ± 1.414	0.358 ± 0.201
Stochastic Petri net	7.31 ± 1.902	$\mathbf{5.504 \pm 2.344}$	8.838 ± 0.649	7.031 ± 0.816	0.359 ± 0.202
	Hospital	**Invoice**	**RTFMP**	**Helpdesk**	
predictive FA (single)	51.689 ± 14.945	1.169 ± 0.06	223.506 ± 74.58	5.13 ± 2.092	
predictive FA (multiple)	72.104 ± 50.779	1.228 ± 0.09	225.467 ± 80.693	$\mathbf{5.046 \pm 2.998}$	
mean FA	42.58 ± 8.622	2.012 ± 0.249	229.337 ± 82.628	5.233 ± 2.022	
adaptive FA	42.58 ± 8.622	1.171 ± 0.061	223.513 ± 78.87	5.233 ± 2.022	
black box (single)	43.234 ± 1.116	1.128 ± 0.242	218.015 ± 38.62	5.802 ± 2.847	
black box (multiple)	39.948 ± 4.314	$\mathbf{1.118 \pm 0.342}$	215.752 ± 70.985	5.746 ± 2.728	
TS (set)	36.74 ± 2.341	1.612 ± 0.375	180.713 ± 71.895	6.039 ± 0.532	
TS (bag)	$\mathbf{36.732 \pm 2.419}$	1.612 ± 0.375	180.713 ± 71.895	5.987 ± 2.519	
TS (sequence)	$\mathbf{36.732 \pm 2.419}$	1.612 ± 0.375	$\mathbf{180.546 \pm 71.277}$	5.987 ± 2.519	
Stochastic Petri net	66.379 ± 29.226	1.664 ± 0.48	190.05 ± 94.548	5.965 ± 0.904	

the MAE of predictions by stochastic Petri net have the lowest volatility among the surveyed methods.

In order to understand why for some of the event logs used mean flow analysis is more accurate than predictive flow analysis, we analyze the performance of these two approaches at the level of individual activities. Specifically, for each activity in the Hospital and the Invoice log, we measure 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 9 , for each activity, we report average MAE of cycle time predictions across all test prefixes. In addition, we report the actual average cycle time values of each activity based on the test set.

As we can see in Table 9 , in the Invoice log, prediction-based cycle times are more accurate than those based on historical averages for the six longest activities which make up the largest portion of the remaining cycle time. Hence, the predictive flow analysis approach provides a better estimation of the overall remaining time than the mean flow analysis approach. In contrast, in the Hospital log, for three of the six longest activities, we get a more accurate estimate using the mean flow analysis. Thus, the performance of the predictive flow analysis approach hinges on the ability to accurately predict cycle times of key activities. In this way, the adaptive approach provides a "golden mean" between the predictive and the mean approaches, while retaining interpretability of the predictions.

To illustrate the interpretability of the proposed white-box techniques, let us consider a partial trace $hd^2(\mathit{SWKD}) = \langle \mathit{NEW}, \mathit{CHANGE\_DIAGN} \rangle$ of an ongoing case SWKD originating from the Hospital event log. At that stage, the case is predicted to exceed the

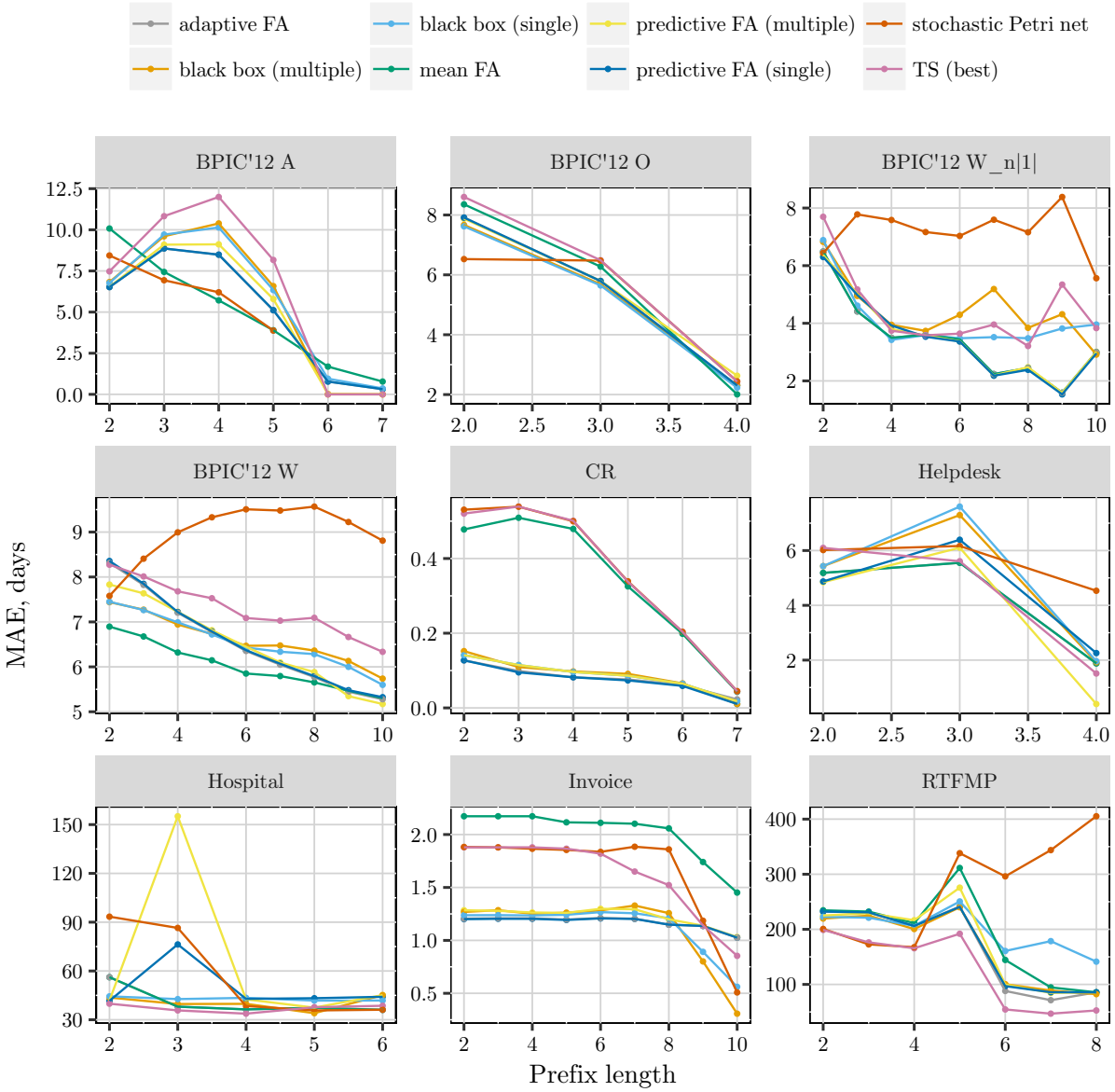

FIGURE 7 Prediction accuracy (measured in terms of MAE) across different prefix lengths.

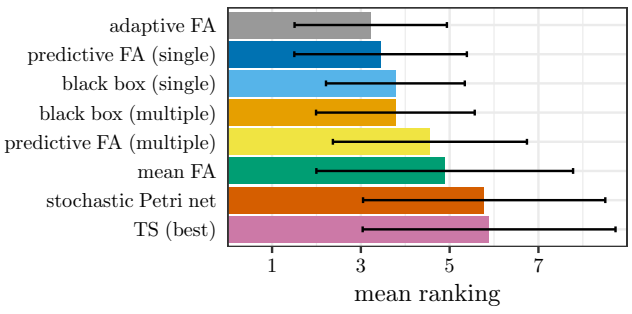

FIGURE 8 Average ranking of the evaluated methods over all datasets. Error bars indicate the 95% confidence interval.

median case duration for the considered process. Our family of flow analysis-based approaches allows users not only to make predictions but also to explain them. Let us consider the output of the adaptive flow analysis approach trained with a single predictive model. Having replayed the trace on the process model (Figure 11), we obtain the following formula for its remaining time:

$$T_{hd^2(SWKD)} = 0 + p2 * (0 + p4 * (CHANGE\_DIAGN)/(1 - p4)) + p5 *$$
$$*(FIN + RELEASE + p7 * CODE\_OK + p8 * CODE\_NOK + BILLED) +$$
$$+p6 * DELETE$$

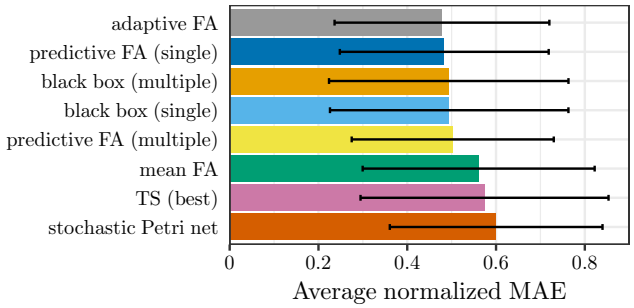

FIGURE 9 Normalized MAE averaged over all prefixes and all datasets.

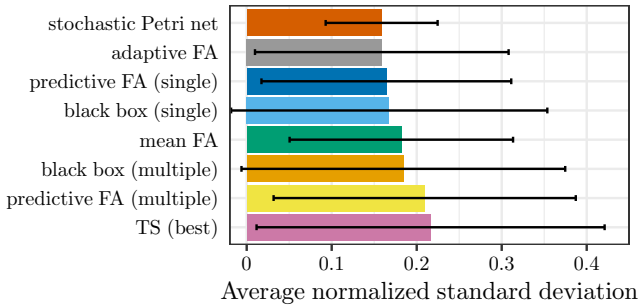

FIGURE 10 Normalized standard deviation of MAE over all prefixes averaged over all datasets.

Table 10 lists the predicted values of cycle times and branching probabilities for the trace in question, as well as their historical averages.

Comparing the obtained values with their historical averages (expectations), we notice that the activity *FIN* is going to take 10 days longer while *BILLED* is about to take 20 days longer. Therefore, by pinpointing *FIN* and *BILLED* as potential bottleneck activities, operational managers may gain more leverage over the process execution and be able to steer this towards the target performance values.

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

Another important observation is that mean flow analysis sometimes outperforms predictive flow analysis. This 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. Nevertheless, the accuracy of the adaptive approach in most logs corresponds to the best accuracy achieved by the predictive and mean methods. In this way, adaptive flow analysis can be regarded as a safeguard against instability in predictions caused by the lack of data (attributes). Thus, we can also answer **RQ2** positively.

TABLE 9 MAE of cycle time predictions of individual activities and their actual mean cycle times (in days).

Activity	MAE		Mean cycle time
	Predictive FA	Mean FA	
Invoice (Top 6 longest activities)			
Invoice_accounting	**2.61**	3.53	3.11
Check_whether_total_approval	**0.771**	0.962	0.893
Manual_identification_CC	**0.109**	0.132	0.097
Compare_of_sums	**0.081**	0.123	0.060
Get_lowest_approval_level	**0.0117**	0.0271	0.0091
Status_change_to_Accounted	**0.0008**	0.0031	0.0005
Hospital			
FIN	**8.52**	58.77	70.08
BILLED	40.36	**36.58**	36.43
DELETE	18.37	**16.76**	8.20
CHANGE_DIAGN	**3.18**	11.91	7.57
CODE_NOK	**14.58**	22.00	6.27
CODE_OK	2.57	**2.38**	5.14

TABLE 10 Predicted and average values of cycle times and branching probabilities for $hd^2(SWKD)$.

Variable	Predicted	Average
p2	0.9988	0.6268
p4	0.0543	0.0305
p5	0.9609	0.9625
p6	0.0391	0.0375
p7	0.9988	0.9766
p8	0.0012	0.0234
CHANGE_DIAGN	1.08	7.57
FIN	80.17	70.08
RELEASE	1.11	1.45
CODE_OK	4.51	5.14
CODE_NOK	7.81	6.27
BILLED	56.21	36.43
DELETE	13.89	8.20

Execution Times

The execution time of the proposed flow analysis approaches 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

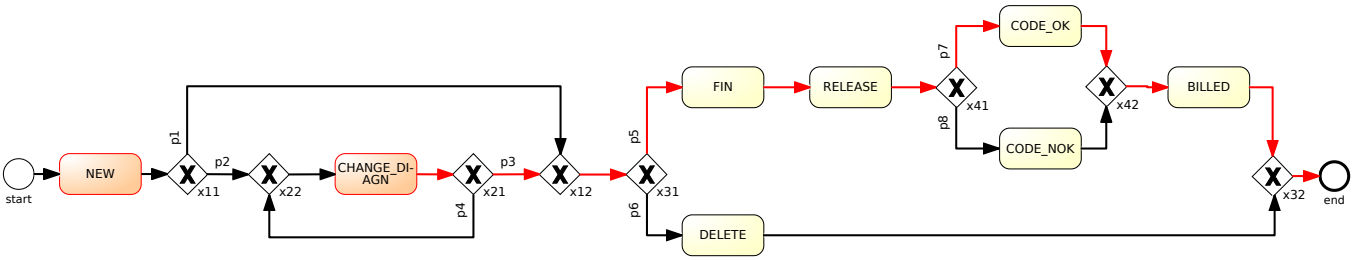

FIGURE 11 A process model of the Hospital log. Current marking of $hd^2(SWKD)$ and its predicted future path are highlighted.

measured the average runtime overhead of our approach. All experiments were conducted on a laptop with a 2.4GHz Intel Core i7 processor and 16GB of RAM.

The training phase includes the time for constructing the prefix log, encoding the prefix traces and fitting the predictive models. It is performed offline and takes between 1 minute (*BPIC'12 O*) and 80 minutes (*Hospital*), depending on the size of the log and the number of models to train, i.e. the number of distinct decision points and activities. Replaying a single test trace takes less than a second. Finally, making the predictions takes between 50 milliseconds and 3 seconds per trace, depending on the length of the trace and the number and complexity of the predictive models. This shows that flow analysis-based prediction approaches perform within reasonable bounds for most online applications.

5.4 | Threats to Validity

The datasets used in this evaluation, except for *BPIC'12 W*, have only completion timestamps, not 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 (XGBoost). With decision trees and random forest, we obtained qualitatively the same results, relative to the baselines. However, our approach is independent of the learning algorithm used. Thus, in principle, using a different algorithm does not 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 family of approaches 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 [51]. A possible enhancement of the proposed approach would be to integrate heuristics that take into account such reliability estimates.

Finally, the degree to which the findings of our study can be generalized is to some extent limited by the fact that the experiments were performed on a limited number of event logs. Even though these are all real-life event logs from various domains that exhibit different properties, it may be possible that the evaluation results would be different on some other event logs. In order to mitigate these threats, we released the source code into the public domain which will allow the replication of our results as well as the repetition of the conducted experiments with other logs.

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 three approaches – one where the formulas' components are predicted from the trace prefix based on the models trained on historical completed traces, another one that instead uses constant values obtained from the historical averages of similar traces, and finally, a hybrid approach that combines the strengths of the above two approaches. We evaluated these three approaches to predict the remaining cycle time, which is a common process performance indicator. The empirical evaluation showed 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 the proposed approaches so

that they would be able to deal with more complex models with overlapping loops, using process model structuring techniques such as the one proposed in [30].

With some modifications in the derivation of the flow analysis formulas, the proposed approaches can be extended to predict other quantitative performance indicators. Another avenue for future work is thus to extend and evaluate the proposed approaches in order to predict cost-related properties in addition to time-related properties.

ACKNOWLEDGMENTS

This research is funded by the Australian Research Council (Grant DP180102839) and the Estonian Research Council (Grant IUT20-55).

References

1. Maggi Fabrizio Maria, Di Francescomarino Chiara, Dumas Marlon, Ghidini Chiara. Predictive monitoring of business processes. In: CAiSE:457–472. Springer; 2014.

2. Leontjeva Anna, Conforti Raffaele, Di Francescomarino Chiara, Dumas Marlon, Maggi Fabrizio Maria. Complex Symbolic Sequence Encodings for Predictive Monitoring of Business Processes. In: BPM:297–313; 2015.

3. Spoel Sjoerd, Keulen Maurice, Amrit Chintan. Process prediction in noisy data sets: a case study in a Dutch hospital. In: International Symposium on Data-Driven Process Discovery and Analysis:60–83. Springer; 2012.

4. Evermann Joerg, Rehse Jana-Rebecca, Fettke Peter. A Deep Learning Approach for Predicting Process Behaviour at Runtime. In: Business Process Management Workshops - BPM 2016 International Workshops, Rio de Janeiro, Brazil, September 19, 2016, Revised Papers:327–338; 2016.

5. Aalst Wil M. P., Schonenberg M. H., Song Minseok. Time prediction based on process mining. *Information systems*. 2011;36(2):450–475.

6. Rogge-Solti Andreas, Weske Mathias. Prediction of remaining service execution time using stochastic Petri nets with arbitrary firing delays. In: ICSOC:389–403. Springer; 2013.

7. Rogge-Solti Andreas, Weske Mathias. Prediction of business process durations using non-Markovian stochastic Petri nets. *Information Systems*. 2015;54:1–14.

8. Dumas Marlon, La Rosa Marcello, Mendling Jan, Reijers Hajo A.. *Fundamentals of Business Process Management, Second Edition*. . Springer; 2018.

9. Verenich Ilya, Nguyen Hoang, La Rosa Marcello, Dumas Marlon. White-box prediction of process performance indicators via flow analysis. In: Proceedings of the 2017 International Conference on Software and System Process, Paris, France, ICSSP 2017, July 5-7, 2017:85–94; 2017.

10. Márquez-Chamorro A. E., Resinas M., Ruiz-Corts A.. Predictive monitoring of business processes: a survey. *IEEE Transactions on Services Computing*. 2017;PP(99):1-1.

11. Pika Anastasiia, Aalst Wil M P, Fidge Colin J, Hofstede Arthur H. M., Wynn Moe T. Predicting deadline transgressions using event logs. In: BPM:211–216. Springer; 2012.

12. Conforti Raffaele, Leoni Massimiliano, La Rosa Marcello, Aalst Wil M. P., Hofstede Arthur H. M.. A recommendation system for predicting risks across multiple business process instances. *Decision Support Systems*. 2015;69:1–19.

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

14. Dongen Boudewijn F, Crooy Ronald A, Aalst Wil M P. Cycle time prediction: when will this case finally be finished?. In: CoopIS:319–336. Springer; 2008.

15. Polato Mirko, Sperduti Alessandro, Burattin Andrea, Leoni Massimiliano. Data-aware remaining time prediction of business process instances. In: 2014 International Joint Conference on Neural Networks, IJCNN 2014:816–823; 2014.

16. Leoni Massimiliano, Aalst Wil M. P., Dees Marcus. A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. *Information Systems*. 2016;56:235–257.

17. Senderovich Arik, Weidlich Matthias, Gal Avigdor, Mandelbaum Avishai. Queue mining for delay prediction in multi-class service processes. *Information systems*. 2015;53:278–295.

18. Senderovich Arik, Di Francescomarino Chiara, Ghidini Chiara, Jorbina Kerwin, Maggi Fabrizio Maria. Intra and Inter-case Features in Predictive Process Monitoring: A Tale of Two Dimensions. In: Lecture Notes in Computer Science, vol. 10445: :306–323. Springer; 2017.

19. Kikas Riivo, Dumas Marlon, Pfahl Dietmar. 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; 2016.

20. Rees-Jones Mitch, Martin Matthew, Menzies Tim. Better Predictors for Issue Lifetime. *CoRR*. 2017;abs/1702.07735.

21. FogBugz . Evidence-Based Scheduling http://help.fogcreek.com/7676/evidence-based-scheduling-ebs. Accessed: 2017-10-23; .

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

23. Breuker Dominic, Matzner Martin, Delfmann Patrick, Becker Jörg. Comprehensible predictive models for business processes. *MIS Quarterly*. 2016;40(4):1009–1034.

24. Tax Niek, Verenich Ilya, La Rosa Marcello, Dumas Marlon. Predictive Business Process Monitoring with LSTM Neural Networks. In: Advanced Information Systems Engineering - 29th International Conference, CAiSE 2017, Essen, Germany, June 12-16, 2017, Proceedings:477–492; 2017.

25. Tan P.N., Steinbach M., Karpatne A., Kumar V.. *Introduction to Data Mining*. What's New in Computer Science Series. Pearson Education; 2013.

26. Polyvyanyy Artem, Smirnov Sergey, Weske Mathias. The Triconnected Abstraction of Process Models. In: Lecture Notes in Computer Science, vol. 5701: :229–244. Springer; 2009.

27. Armas-Cervantes Abel, Baldan Paolo, Dumas Marlon, García-Bañuelos Luciano. Diagnosing behavioral differences between business process models: An approach based on event structures. *Information systems*. 2016;56:304–325.

28. Aalst Wil M. P.. Process Discovery: Capturing the Invisible. *IEEE Comp. Int. Mag.*. 2010;5(1):28–41.

29. Augusto Adriano, Conforti Raffaele, Dumas Marlon, et al. Automated Discovery of Process Models from Event Logs: Review and Benchmark. *CoRR*. 2017;abs/1705.02288.

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

31. Teinemaa Irene, Dumas Marlon, Maggi Fabrizio Maria, Di Francescomarino Chiara. Predictive Business Process Monitoring with Structured and Unstructured Data. In: Business Process Management - 14th International Conference, BPM 2016, Rio de Janeiro, Brazil, September 18-22, 2016:401–417; 2016.

32. Mitchell Tom M.. *Machine learning*. McGraw-Hill; 1997.

33. Fayyad Usama M., Piatetsky-Shapiro Gregory, Smyth Padhraic, Uthurusamy Ramasamy, eds. *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press; 1996.

34. Chen Tianqi, Guestrin Carlos. XGBoost: A Scalable Tree Boosting System. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016:785–794; 2016.

35. Fowler B., Rajendiran M., Schroeder T., Bergh N., Flower A., Kang H.. Predicting patient revisits at the University of Virginia Health System Emergency Department. In: 2017 Systems and Information Engineering Design Symposium (SIEDS):253-258; 2017.

36. Möller A, Ruhlmann-Kleider V, Leloup C, et al. Photometric classification of type Ia supernovae in the SuperNova Legacy Survey with supervised learning. *Journal of Cosmology and Astroparticle Physics*. 2016;2016(12):008.

37. Torlay L., Perrone-Bertolotti Marcela, Thomas E., Baciu Monica. Machine learning-XGBoost analysis of language networks to classify patients with epilepsy. *Brain Informatics*. 2017;4(3):159–169.

38. Olson Randal S., La Cava William, Mustahsan Zairah, Varik Akshay, Moore Jason H.. Data-driven Advice for Applying Machine Learning to Bioinformatics Problems. *CoRR*. 2017;abs/1708.05070.

39. Hastie Trevor, Tibshirani Robert, Friedman Jerome H.. *The elements of statistical learning: data mining, inference, and prediction, 2nd Edition.* . Springer; 2009.

40. Urraca-Valle Ruben, Antoñanzas Javier, Antoñanzas-Torres Fernando, Pisón Francisco Javier. Estimation of Daily Global Horizontal Irradiation Using Extreme Gradient Boosting Machines. In: International Joint Conference SOCO'16-CISIS'16-ICEUTE'16 - San Sebastián, Spain, October 19th-21st, 2016, Proceedings:105–113; 2016.

41. Bergstra James, Bengio Yoshua. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*. 2012;13:281–305.

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

43. Andrews Robert, Suriadi Suriadi, Wynn Moe, et al. Comparing static and dynamic aspects of patient flows via process model visualisations. *Preprint available at https://eprints.qut.edu.au/102848/*. 2016;.

44. Raudys Sarunas, Jain Anil K.. Small Sample Size Effects in Statistical Pattern Recognition: Recommendations for Practitioners. *IEEE Trans. Pattern Anal. Mach. Intell.*. 1991;13(3):252–264.

45. Teinemaa Irene, Dumas Marlon, La Rosa Marcello, Maggi Fabrizio Maria. Outcome-Oriented Predictive Process Monitoring: Review and Benchmark. *CoRR*. 2017;abs/1707.06766.

46. Ranzato Marc'Aurelio, Huang Fu Jie, Boureau Y-Lan, LeCun Yann. Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition. In: 2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007), 18-23 June 2007, Minneapolis, Minnesota, USA; 2007.

47. Hyndman Rob J, Koehler Anne B. Another look at measures of forecast accuracy. *International Journal of Forecasting.* 2006;22(4):679–688.

48. Pedregosa F., Varoquaux G., Gramfort A., et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research.* 2011;12:2825–2830.

49. Fukunaga Keinosuke, Hayes Raymond R.. Effects of Sample Size in Classifier Design. *IEEE Trans. Pattern Anal. Mach. Intell.*. 1989;11(8):873–885.

50. Tabachnick Barbara G. *Using multivariate statistics: Sas Workbook*. Addison-Wesley; 1996.

51. Kull Meelis, Flach Peter A.. 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; 2014.

How to cite this article: I. Verenich, H. Nguyen, M. Dumas, M. La Rosa, and A. ter Hofstede (2017), Predicting Process Performance: A White-Box Approach, *Journal of Software: Evolution and Process*, *2017;00:1–6*.