A Data-adaptive Trace Abstraction Approach to the Prediction of Business Process Performances

Antonio Bevacqua[1], Marco Carnuccio[1], Francesco Folino[2], Massimo Guarascio[2] and Luigi Pontieri[2]
[1]*DIMES Department, University of Calabria, via P. Bucci 41C, 87036, Rende, CS, Italy*
[2]*ICAR-CNR, National Research Council of Italy, via P. Bucci 41C, 87036, Rende, CS, Italy*

Keywords: Data Mining, Regression, Clustering, Business Process Analysis.

Abstract: This paper presents a novel approach to the discovery of predictive process models, which are meant to support the run-time prediction of some performance indicator (e.g., the remaining processing time) on new ongoing process instances. To this purpose, we combine a series of data mining techniques (ranging from pattern mining, to non-parametric regression and to predictive clustering) with ad-hoc data transformation and abstraction mechanisms. As a result, a modular representation of the process is obtained, where different performance-relevant variants of it are provided with separate regression models, and discriminated on the basis of context information. Notably, the approach is capable to look at the given log traces at a proper level of abstraction, in a pretty automatic and transparent fashion, which reduces the need for heavy intervention by the analyst (which is, indeed, a major drawback of previous solutions in the literature). The approach has been validated on a real application scenario, with satisfactory results, in terms of both prediction accuracy and robustness.

1 INTRODUCTION

The general aim of process mining techniques (van der Aalst et al., 2003) is to extract information from historical log "traces" of a business process (i.e., sequences of events registered during different enactments of the process), in order to help analyze and possibly improve it. An emerging trend in this field (see, e.g., (van Dongen et al., 2008; van der Aalst et al., 2011; Folino et al., 2012)) concerns the prediction of performance indicators (defined on each process instance), as a way to help improve future process enactments, through, e.g., recommendation or risk analysis. In general, these approaches face the problem by inducing some kind of prediction model for the given performance indicator, based on some suitable trace abstraction function (mapping, e.g., the trace onto the set/multiset of process tasks appearing in it). In particular, a non-parametric regression model is used in (van Dongen et al., 2008) to build the prediction for a new (possibly partial) trace, based on its similarity towards a set of historical ones – where the similarity between two traces is evaluated by comparing their respective abstract views. However, such an instance-based scheme is likely to take long prediction times (unsuitable for many real run-time environments), especially in the case of complex and flexible processes, where a large amount of historical traces should be kept (and retrieved) to capture adequately its wide range of behaviors. A model-based prediction scheme is conversely followed in (van der Aalst et al., 2011), where an annotated finite-state machine (FSM) model is induced from the input log, with the states corresponding to abstract representation of log traces. The discovery of such FSM models was combined in (Folino et al., 2012) with a context-driven (predictive) clustering approach, in a way that different execution scenarios can be discovered for the process, and equipped with distinct (more specific and more precise) local predictors.

In general, a critical issue in the induction of such prediction models, especially in the case of complex business processes, concerns the definition of a suitable trace abstraction function, capable to focus on the core properties of the events (happened in a process instance) that impact the more on its performance outcomes. In fact, as discussed in (van der Aalst et al., 2011), choosing the right abstraction level is a delicate task, where an optimal balance has to be reached between the risks of overfitting (i.e., having an overly detailed model, nearly replicating the training set, which will hardy provide accurate forecasts over unseen cases) and of undefitting (i.e., the model

Bevacqua A., Carnuccio M., Folino F., Guarascio M. and Pontieri L..
A Data-adaptive Trace Abstraction Approach to the Prediction of Business Process Performances.
DOI: 10.5220/0004448700560065
In *Proceedings of the 15th International Conference on Enterprise Information Systems* (ICEIS-2013), pages 56-65
ISBN: 978-989-8565-59-4

is too abstract and imprecise, both on the training cases and on new ones).

Previous approaches mainly leave the responsibility to tune the abstraction level to the analyst, allowing her/him to select the general form of trace abstractions (e.g., set/multiset/lists of tasks), and to possibly fix a maximal horizon threshold h — i.e., only the properties of the h more recent events in a trace can be used in its abstract view, so discarding older events. On the other hand, FSM-like models cannot effectively exploit non-structural (context-oriented) properties of the process instances, which might actually impact on performances as well. In fact, the idea of including such data (in addition to tasks) in the construction of trace abstractions will clearly emphasize the combinatorial explosion issue discussed above.

Core Idea and Contributions. We try to overcome the above limitations, by devising a novel approach, capable of both taking full advantage of "non structural" context data, and of finding a good level of abstraction over on the history of process instances, in a pretty automated and transparent fashion.

Our core belief is that handy (and yet accurate enough) prediction models can be learnt via various existing model-based regression algorithms (either parametric, such as, e.g., (Hardle and Mammen, 1993; Quinlan, 1992), or non-parametric, such as, e.g., (Harlde, 1990; Witten and Frank, 2005)), rather than resorting to an explicit representation of process states (like in (van der Aalst et al., 2011; Folino et al., 2012)) or to an instance-based approach, like in (van Dongen et al., 2008). This clearly requires that an adequate propositional representation of the given traces is preliminary build, capturing both structural (i.e., task-related) and ("non-structural") aspects. To this end, we propose to convert each process trace into a set or a multi-set of process tasks, and let the regression method decide automatically which and how the basic structural elements in such an abstracted view of the trace are to be used to make a forecast.

Moreover, we still leverage the idea of (Folino et al., 2012), of combining performance prediction with a predictive clustering technique (Blockeel and Raedt, 1998), in order to distinguish heterogeneous context-dependent execution scenarios ("variants") for the analyzed process, and eventually provide each of them with a specialized regressor. In fact, such an approach brings, in general, substantial gain in terms of readability and accuracy, besides explicitly showing the dependence of discovered clusters on context features, and speeding up (and possibly parallelize) the computation of regression models — which are typically more compact, more precise and easier to read/evaluate/validate than a single regression model extracted out of the whole log. In fact, we believe that (as confirmed by the empirical results in Section 5) even very simple regression methods can furnish robust and accurate predictions, if combined with a properly devised clustering procedure. Moreover, this can even allow for a scalable usage of instance-based regression schemes (which are likely to take prohibitive prediction times on real logs), seeing as only the traces of a single cluster (selected on the basis of context features) would be scanned over.

We pinpoint that the target features used in the clustering (where context features conversely act as descriptive attributes) are derived from frequent structural patterns (still defined as sets or bags of tasks), instead of directly using the abstract representations extracted by the log, as done in (Folino et al., 2012). These patterns will be discovered efficiently via an ad-hoc *a-priori*-like (Agrawal and Srikant, 1994) method, where the analyst is allowed to specify a minimum support threshold, and possibly an additional ("gap") constraint, both enforced in the very generation of the patterns. Notably, such an approach frees the analyst from the burden of explicitly setting the abstraction level (i.e., the size of patterns, in our case), which is determined instead in a data-driven way.

Organization. The rest of the paper is structured as follows. After introducing some handy notation and core concepts in Section 2, we present the proposed approach, in an algorithmic form, in Section 3. We then discuss the implementation of the method in Section 4, and an experimentation on a real data in Section 5. Section 6 finally presents some concluding remarks and future work directions.

2 FORMAL FRAMEWORK

2.1 Logs and Performances

As usually done in the literature, we assume that for each process instance (a.k.a "case") a *trace* is recorded, storing the sequence of *events* happened during its unfolding. Let $\mathcal{T}$ be the universe of all (possibly partial) traces that may appear in any log of the process under analysis. For any trace $\tau \in \mathcal{T}$, $len(\tau)$ is the number of events in τ, while $\tau[i]$ is the i-th event of τ, for $i = 1 \,..\, len(\tau)$, with $task(\tau[i])$ and $time(\tau[i])$ denoting the task and timestamp of $\tau[i]$, respectively. We also assume that the first event of each trace is always associated with a unique "initial" task A_0 (possibly added artificially before analyzing the log), and

its timestamp registers the time when the corresponding process instance started.

Let us also assume that, for any trace τ, a tuple $context(\tau)$ of data is stored in the log to keep information about the execution context of τ — like in (Folino et al., 2012), this tuple may gather both data properties and environmental features (characterizing the state of the BPM system). For ease of notation, let $\mathcal{A}^{\mathcal{T}}$ denote the set of all the tasks (a.k.a., activities) that may occur in some trace of $\mathcal{T}$, and $context(\mathcal{T})$ be the space of context vectors — i.e., $\mathcal{A}^{\mathcal{T}} = \cup_{\tau \in \mathcal{T}} tasks(\tau)$, and $context(\mathcal{T}) = \{context(\tau) \mid \tau \in \mathcal{T}\}$.

Further, $\tau(i]$ is the *prefix* (sub-)trace containing the first i events of a trace τ and the same context data (i.e., $context(\tau(i] = context(\tau)$), for $i = 0 \,..\, len(\tau)$.

A *log* L is a finite subset of $\mathcal{T}$, while the *prefix set* of L, denoted by $\mathcal{P}(L)$, is the set of all the prefixes of L's traces, i.e., $\mathcal{P}(L) = \{\tau(i] \mid \tau \in L \text{ and } 1 \leq i \leq len(\tau)\}$.

Let $\hat{\mu} : \mathcal{T} \rightarrow \mathbb{R}$ be an (unknown) function assigning a performance value to any (possibly unfinished) process trace. For the sake of concreteness, we will focus hereinafter on the special case where the target performance value associated with each trace is the remaining process time (measured, e.g., in days, hours, or in finer grain units), i.e., the time needed to finish the corresponding process enactment. Moreover, we assume that performance values are known for all prefix traces in $\mathcal{P}(L)$, for any given log L. In fact, for each trace τ, the (actual) remaining-time value of $\tau(i]$ is $\hat{\mu}(\tau(i]) = time(\tau[len(\tau)]) - time(\tau[i])$.

A (predictive) *Process Performance Model (PPM)* is a model that can estimate the unknown performance value (i.e., the remaining time in our setting) of a process enactment, based on the contents of the corresponding trace. Such a model can be viewed as a function $\mu : \mathcal{T} \rightarrow \mathbb{R}$ estimating $\hat{\mu}$ all over the trace universe — which also includes the prefix traces of all possible unfinished enactments of the process. Learning a PPM hence amounts to solving a particular induction problem, where the training set takes the form of a log L, and the value $\hat{\mu}(\tau)$ of the target measure is known for each (sub-)trace $\tau \in \mathcal{P}(L)$.

Recent approaches to this problem (van der Aalst et al., 2011; van Dongen et al., 2008; Folino et al., 2012) leverage all the basic idea of applying some suitable abstraction function to process traces, with the ultimate aim of capturing only those facets of the registered events that influence the most process performances — while disregarding minor details.

2.2 Trace Abstraction

An abstracted (structural) view of a trace gives a concise description of the tasks executed during the corresponding process enactment. Two common ways of building such a view consist in simply regarding the trace as a multiset (a.k.a. bag) or as a set of tasks, are formally defined below.

Definition 1 (Structural Trace Abstraction)**.** Let $\mathcal{T}$ be a trace universe and $A_0, \ldots, A_n$ be the tasks in $\mathcal{A}^{\mathcal{T}}$. A *structural (trace-) abstraction function* $struct^{mode} : \mathcal{T} \rightarrow \mathcal{R}^{mode}_{\mathcal{T}}$ is a function mapping each trace $\tau \in \mathcal{T}$ to an an *abstract representation* $struct^{mode}(\tau)$, taken from a *abstractions' space* $\mathcal{R}^{mode}_{\mathcal{T}}$. Two concrete instantiations of the above function, denoted by $struct^{bag} : \mathcal{T} \rightarrow \mathbb{N}^n$ (resp., $struct^{set} : \mathcal{T} \rightarrow \{0,1\}^n$), are defined next to which map each trace $\tau \in \mathcal{T}$ onto a bag-based (resp., set-based) representation of its structure: *(i)* $struct^{bag}(\tau) = \langle count(A_0, \tau), \ldots, count(A_n, \tau) \rangle$, where $count(A_i, \tau)$ is the number of times that task A_i occurs in τ; and *(ii)* $struct^{set}(\tau) = \langle occ(A_0, \tau), \ldots, occ(A_n, \tau) \rangle$, where $occ(A_i, \tau) = \texttt{true}$ iff $count(A_i, \tau) > 0$. □

The two concrete abstraction "modes" (namely, *bag* and *set*) defined above summarize any trace τ into a vector, where each component corresponds to a single process task A_i, and stores either the number of times that A_i appears in the trace τ, or (respectively) a boolean value indicating whether A_i occur in τ or not. Notice that, in principle, we could define abstract trace representations as sets/bags over another property of the events (e.g., the executor, instead of the task executed), or even over a combination of event properties (e.g., the task plus who performed it).

Example 1. Let us consider a real-life case study pertaining a transshipment process, used for the experiments described in Section 5. Basically, for each container c passing through the harbor, a distinct log trace τ_c is stored, registering all the tasks applied to c, which may include: moving c by means of either a straddle-carrier (MOV), swapping c with another container (SHF), and loading c onto a ship by way of a shore crane (OUT). Let τ be a log trace storing a sequence $\langle e_1, e_2, e_3 \rangle$ of three events such that $task(e_1) = task(e_2) = MOV$ and $task(e_3) = OUT$. With regard to the abstract trace representations introduced in Def. 1, it is easy to see that $struct^{bag}(\tau) = [2,0,1]$, and $struct^{set}(\tau) = [1,0,1]$ — where the traces are mapped into a vector space consisting of the dimensions $A_1 \equiv MOV$, $A_2 \equiv SHF$, $A_3 \equiv OUT$. ◁

The structural abstraction functions in Def. 1 are a subset of the ones used in previous approaches to the discovery of predictive process models (van der Aalst et al., 2011; van Dongen et al., 2008; Folino et al., 2012). To be more precise, (van Dongen et al., 2008) also considers the possibility to map a trace into a vector of task durations, as well as to combine mul-

tiple structural abstractions with data attributes of the traces, while the other two approaches also allow for abstracting a trace into the list of tasks appearing in it (as an alternative to bag/set -oriented abstractions).

2.3 Clustering-based PPM

Like in (Folino et al., 2012), we here consider a special kind of PMM model, relying on a clustering of the process traces. Such a model, described below, is indeed a predictive clustering model, where context data play the role of descriptive attributes, whereas the target variables are derived by specific performance values, extracted out of the traces.

Definition 2 (Clustering-based Performance Prediction Model (`CB-PPM`))**.** Let L be a log (over $\mathcal{T}$), with context features $context(\mathcal{T})$, and $\hat{\mu} : \mathcal{T} \to \mathbb{R}$, be a performance measure, known for all $\tau \in \mathcal{P}(L)$. Then a *clustering-based performance prediction model* (`CB-PPM`) for L is a pair $M = \langle c, \langle \mu_1, \ldots, \mu_k \rangle \rangle$, which encodes the unknown performance function $\hat{\mu}$ in terms of a predictive clustering model, with k denoting the associated number of clusters (found for L). More specifically, c is a partitioning function, which assigns any (possibly novel) trace to one of the clusters (based on its context data), while each μ_i is the *PPM* of the i-th cluster — i.e., $c : context(\mathcal{T}) \to \{1, \ldots, k\}$, and $\mu_i : \mathcal{T} \to \mathbb{R}$, for $i \in \{1, \ldots, k\}$. The performance $\hat{\mu}(\tau)$ of any (partial) trace τ is eventually estimated as $\mu_j(\tau)$, where j=$c(context(\tau))$. □

In this way, each cluster has its own PPM model, encoding how $\hat{\mu}$ depends on the structure (and, possibly, the context) of a trace, relatively that cluster. The prediction for each trace is hence made with the predictor of the cluster it is assigned to (by function c).

In general, such an articulated kind of PPM can be built by inducing both a predictive clustering model and multiple PPMs (as the building blocks implementing c and all μ_i, respectively). In particular, in (Folino et al., 2012), the latter task is accomplished by using the method in (van der Aalst et al., 2011), so that each cluster is eventually provided with an A-FSM model. As mentioned in Section 1, in order to develop an easier-to-use and data-adaptive approach, we will not use A-FSM models (which typically require a careful explicit setting of the abstraction level), and will rather employ one of the various regression methods available for propositional data. To this purpose, an ad-hoc view of the log will be produced, where both the context-oriented and structure-oriented (cf. Def. 1) features of a trace are used as descriptive attributes, whereas the target attributes are derived by projecting the trace onto a space of structural patterns. These patterns, which will be computed via an ad-hoc data mining method, are described in detail in the following.

2.4 Structural Patterns

In our setting, structural patterns are meant to capture regularities in the structure of traces, abstracted via sets or bags of tasks. In particular, these patterns can be regarded as (constrained) sub-sets or sub-bag of tasks that appear frequently in the (abstracted) log traces. Let $mode \in \{bag, set\}$ denotes a given abstraction criterion, $\mathcal{T}$ be the reference trace universe, and $\mathcal{A}^{\mathcal{T}} = \{A_0, \ldots, A_n\}$ be its associated process tasks. Then, a *(structural) pattern* w.r.t. $\mathcal{T}$ and *mode* simply is an element p of the *abstractions' space* $\mathcal{R}_{\mathcal{T}}^{mode}$ – over which the structural trace-abstraction function $struct^{mode}$ ranges indeed. The size of p, denoted by $size(p)$, is the number of distinct tasks in p (i.e., the number of p's components with a positive value.

Having a structural pattern the same form as a (structural) trace abstraction, we can apply usual set/bag containment operators to them both. Specifically, given two elements p and p' of $\mathcal{R}_{\mathcal{T}}^{mode}$ (be each a pattern or a full trace representation), we say that p_2 *contains* p_1 (and symmetrically p_1 is contained in p_2), denoted by $p_1 \preceq p_2$, if $p_1[j] \leq p_2[j]$, for $j = 1, \leq, n$, and for $i = 1, 2$ — where $p_i[j]$ is the i-th component of p_i, viewed as a vector in $\mathcal{D}^n$ (cf. Def. 1).

As we want to eventually use such patterns for clustering purposes, we are interested in those that capture significant behavioral schemes. In particular, an important property required for such patterns is that they occur frequently in the given log (otherwise, little, low significant clusters are likely to be discovered), as specified in the following notion of support.

Definition 3 (Pattern Support and Footprints)**.** Let $\tau \in \mathcal{T}$ be a trace, *mode* be a given abstraction mode, and p be pattern (w.r.t. $\mathcal{T}$ and *mode*).

Then, we say that τ *supports* p, denoted by $\tau \vdash p$ if its corresponding structural abstraction contains p (i.e., $p \preceq struct^{mode}(\tau)$). In such a case, a *footprint* of p on τ is subset $F = \{f_1, \ldots, f_k\} \subseteq \{1, \ldots, len(\tau)\}$ of positions within τ, such that $struct^{mode}(\langle \tau[f_1], \ldots, \tau[f_k] \rangle) = p$. Moreover, $gap(F)$ is the number of events in τ which match no position of F and appear in between a pair of matching events — i.e., $gap(F) = \max_{f_i \in F}\{f_i\} - \min_{f_i \in F}\{f_i\} - |F| + 1$. Finally, with little abuse of notation, we denote $gap(p, \tau) = \min\ \{\{\infty\} \cup \{\ gap(F) \mid F \text{ is a footprint of } p \text{ on } \tau\ \}\}$. □

In words, a footprint F of a pattern p, on a trace τ supporting it, identifies a subsequence of τ which (i) contains the events occurring in τ at one of the positions in F, and (ii) has a structural representation

Input: A log L over a trace universe $\mathcal{T}$, with associated tasks $AS = A_1, \ldots, A_n$ and target performance measure $\hat{\mu}$ (known over $\mathcal{P}(L)$), an abstraction mode $m \in \{set, bag\}$ (cf. Def. 1), three thresholds, $minSupp \in [0,1)$, $maxGap \in \mathbb{N} \cup \{\infty\}$, and $K_{top} \in \mathbb{N}^+ \cup \{\infty\}$, and a base regression method $REGR$

Output: A `CB-PPM` model for L (fully encoding $\hat{\mu}$ all over $\mathcal{T}$).

Method: Perform the following steps:

1 Let $context(\tau)$ be the vector of context data associated with each $\tau \in L$;
2 Build a *structural view* S_L of $\mathcal{P}(L)$, by replacing each $\tau \in \mathcal{P}(L)$ with a transaction-like representation of $struct^m(\tau)$;
3 $RSP :=$ `minePatterns`$(S_L, m, minSupp, maxGap)$;
4 $RSP :=$ `filterPatterns`$(RSP, kTop)$;
6 Let $RSP = \{p_1, \ldots, p_s\}$;
7 Build a *log sketch* P_L for L, by using both context data and RSP-projected performances;
8 Learn a PCT T, using $context(\tau)$ (resp., $val(\tau, p_i)$, i=1..s) as descriptive (resp., target) features for each $\tau \in L$;
9 Let $L[1], \ldots, L[k]$ denote the discovered clusters;
10 **for each** $L[i]$ **do**
11 Induce a regression model ppm_i out of $\mathcal{P}(L[i])$, using method $REGR$ — regarding, for each $\tau \in \mathcal{P}(L[i])$, $context(\tau)$ and $struct^m(\tau)$ as the input values, and the performance measurement $\hat{\mu}(\tau)$ as the target value;
10 Store ppm_i as the implementation of the prediction function $\mu_i : \mathcal{T} \rightarrow M$ (for cluster i);
11 **end**
12 **return** $\langle c, \{\mu_1, \ldots, \mu_k\} \rangle$.

Figure 1: **Algorithm** `AA-PPM Discovery`.

coinciding with p. Clearly, if p is not supported by τ, $gap(p, \tau)$ will take an infinite value.

3 SOLUTION ALGORITHM

Figure 1 illustrates the main steps of our approach to the discovery of a CB-PPM model, in the form of an algorithm, named `AA-PPM Discovery`. Essentially, the problem is approached in three main phases.

In the first phase (Steps 1-5), a set of (frequent) structural patterns are extracted from the log, which are deemed to capture the main behavioral schemes of the process, as concerns the dependence of performance on the execution of tasks. To this end, after converting the structure of each (possibly partial) trace τ into an itemset (Step 2) [1], we compute all the structural patterns (i.e., sub-sets, of various sizes) that occur frequently in the log and effectively summarize the behaviors in the log. More precisely, we first compute the set $\{p \in \mathcal{R}^m_{\mathcal{T}} \mid supp^{maxGap}(p, S_L) \geq minSupp\}$ (cf. Def.3), by using function `minePatterns`, which is stored in RSP — note that this set will never be empty, since (as an extreme case) at least a singleton pattern with A_0 is frequent (no matter of $minSupp$, m and $maxGap$). These patterns are then filtered by function `filterPatterns`, which selects the $kTop$ most relevant patterns among them. Notably, we can still use all the discovered patterns, by fixing $maxGap = \infty$ (no real filter is applied in this case). Both these functions are explained in details later on.

In the second phase, the selected patterns are used to associate a series of numerical variables with all traces (Step 7), and to carry out a predictive clustering of them (Step 8). To this end, a propositional view of the log, here named *log sketch*, is produced by transforming each trace into a tuple, where context properties play as are descriptive attributes and the projection onto the space of selected patterns are the target numerical features. Specifically, any selected pattern p gives rise to a target (performance) feature, such that the value $val(\tau, p)$ taken by it on any trace τ is be computed as follows: **(i)** $val(\tau, p) =$ `NULL`, if $\tau \nvdash p$, or **(ii)** $val(\tau, p) = \hat{\mu}(\tau(j^*])$, where j^* is the biggest index $j \in \{1, \ldots, len(\tau)\}$ such that $\tau(j] \vdash p$. Like in (Folino et al., 2012), the clustering is computed by inducing a Predictive Clustering Tree (PCT) (Blockeel and Raedt, 1998) from the log sketch (Step 8).

Finally, each cluster is equipped with a basic (not clustering-based) PPM model, by using some suitable regression method (chosen through parameter $REGR$), provided with a dataset encoding all the prefixes that can be derived from the traces assigned to the cluster. Specifically, each such prefix τ is encoded as a tuple where $context(\tau)$ and $struct^m(\tau)$ are regarded as input values, while the associated performance measurement $\hat{\mu}(\tau)$ represents the value of the numerical target variable that is to be estimated.

3.1 Function `minePatterns`

This function computes all the patterns, of any size, that get a support equal to or higher than $minSupp$

[1] In particular, as to bags, any $s = struct^{bag}(\tau) \in N^n$ is turned into $\{(A_i, k_j) \mid 0 \leq i \leq n, s[i] > 0 \text{ and } 1 \leq j \leq s[i]\}$.

in a transactional view of trace structures. Notably, the function does not require the analyst to specify the size of each pattern (differently from the horizon threshold h used in previous methods), which is instead determined automatically, in a data-driven way. However, it allows for possibly fixing a finite *maxGap* threshold for the gaps admitted between patterns and traces, in case she/he wants to keep some more details on the actual sequencing of the tasks. Technically, this constraint is used to introduce a refined support function, defined as follows:

$$supp^{maxGap}(p,L) = |\{\tau \in L \mid \tau \vdash p \text{ and } gap(p,\tau) < maxGap+1\}| \quad (1)$$

Note that this function actually coincides with the standard one when $maxGap = \infty$ (under the usual convention that $\infty + 1 = \infty$). As a result, `minePatterns` will return each pattern $p \in \mathcal{R}_{\mathcal{T}}^{m}$ such that $supp^{maxGap}(p,L) \geq minSupp$. It can be proved that this computation can be done in a level-wise way, despite the fact that the support function does not enjoy any (anti-)monotonicity property.

The basic computation scheme for the function `minePatterns`, sketched in Figure 2, assumes that: m is the chosen (structural) trace abstraction mode, *minSupp* and *maxGap* are the thresholds for specifying support and gap requirements, respectively, and L is the original log, taken from a reference trace universe $\mathcal{T}$. In practice, the function actually works with a transactional encoding of L, storing a set-oriented representation of $struct^{m}(\tau)$ for each $\tau \in \mathcal{P}(L)$).

Function `minePatterns` (S_L: transaction set; *minSupp*: real; *maxGap*: integer; $m : \{set, bag\}$): a set of frequent structural patterns (i.e., a subset of $\mathcal{R}_{\mathcal{T}}^{m}$)

$I_1 := \{x \in AS \mid minSupp \times |S_L| \leq |\{t \in S_L \mid x \text{ is in } t\}|\}$;
$I_k := I_1$;
$\mathcal{F} := I_1$;
while $I_k \neq \emptyset$ **do**
  $C \leftarrow \{p' \mid p' = p \cup \{x\} \wedge p \in I_k \wedge x \in I_1 \wedge x \notin p\}$;
  for each $\tau \in \mathcal{P}(L)$ **do**
    $C_\tau \leftarrow \{p \mid p \in C \wedge \tau \vdash p \wedge gap(p,\tau) < maxGap+1\}$
    for each $p \in C_\tau$ **do**
      $count[p] \leftarrow count[p] + 1$;
    end
  end
  $I_k := \{ p \mid p \in C \wedge count[p] \geq minSupp \times |S_L| \}$;
  $\mathcal{F} := \mathcal{F} \bigcup I_k$;
end
return $\mathcal{F}$

Figure 2: **Function** `minePatterns`.

Clearly, all the above constraints (including the gaps one) are enforced at each level of the pattern generation procedure, with the advantage of shrinking the amount of patterns generated at each step (k), as well as the overall computation time.

This can be done with no risk of endangering the correctness and completeness of results, as informally proven in the folllow.

Let us first consider the case $maxGap = \infty$. As, in this case, function $supp^{maxGap}$ (cf. Eq. 3.1) is anti-monotonic w.r.t. pattern size, when computing the patterns of size k we can safely filter out any pattern p of them such that $supp(p) < minSupp$ – indeed, there cannot be any pattern p' such that $size(p') = k+1$, $p \preceq p'$ and $supp(p') \geq minSupp$.

By converse, such a monotonicity property is not enjoyed by gap constraints. Indeed, when removing an element from a pattern, the resulting pattern may contain a higher number of spurious activities (i.e., may get a higher gap score) on some traces. For example, consider the pattern $p = \{a,b,c,d\}$ (of size 4) and the trace τ encoding the sequence $y_1,\ldots,y_q,\mathbf{a},\mathbf{b},x_1,x_2,\mathbf{c},x_3,\mathbf{d},z_1,\ldots,z_s$ of task labels. For the sake of simplicity, and w.l.o.g., let us assume that the tasks a,b,c,d do not occur in any other parts of the traces (i.e., $\{a,b,c,d\} \cap \{y_1,\ldots,y_q,z_1,\ldots,z_s)\} = \emptyset$), so that we can focus on the subsequence of tasks $\mathbf{a},\mathbf{b},x_1,x_2,\mathbf{c},x_3,\mathbf{d}$ — which actually determines the gap score for p and τ.

Clearly, for every pattern p' obtained by removing an "internal" element (i.e., different from both a and d in the example) from a given pattern p, it holds $gap(p',\tau) \geq gap(p,\tau)$ – e.g., $gap(\{a,c,d\},\tau) = gap(\{a,b,d\},\tau) = 4 > 3 = gap(p,\tau)$.

However, the two patterns obtained by removing one of the two "extreme elements" (i.e., either a or d) are guaranteed to have the same gap as p, or even to lower it. For example, $gap(\{b,c,d\},\tau) = 3 = gap(p,\tau)$, and $gap(\{a,b,c\},\tau) = 2 < gap(p,\tau)$.

Consequently, in the computation scheme depicted above, for any relevant pattern p of level $k > 1$ (i.e., for any k-sized pattern p such that $supp^{maxGap}(p,L) \geq minSupp$), there are at least two relevant sub-patterns of level $k-1$ which will produce p when merged together.

3.2 Function `filterPatterns`

The function is meant to select a subset of mostly significant and useful patterns, in order to allow for a more effective and more scalable computation of predictive clustering models. In particular, we want to prevent the case where the PCT learning algorithm has to work in a sparse and high-dimensional target space, where low-quality will hardly be found, while long computation times are likely to take place.

Hence, we allow the analyst to ask for only keeping the *kTop* patterns that seems to discriminate the main performance profiles at best. To this end, we employ a variant of the scoring function ϕ proposed in (Folino et al., 2012) (giving score 0 to every feature with no positive correlation with context data), which is essentially meant to give preference to patterns ensuring a higher values of the following measures: (i) support, (ii) correlation with the context attributes and (iii) and variability of the associated performance values (i.e., $val(p,\tau)$, with τ ranging over L). Further details can be found in (Folino et al., 2012).

4 IMPLEMENTATION

The prototype system AA-TP (Adaptive-Abstraction Time Prediction), specializes algorithm *AA-PPM Discovery* to the case where the remaining processing time is the target performance. The logical architecture of the system is sketched in Figure 3, where the arrows between blocks stand for information flows, while *Log Data* is a collection of process logs represented in the MXML (or XES) format (van der Aalst and *et al.*, 2007).

The *Scenario Discovery* module is responsible for identifying behaviorally homogeneous groups of traces, in terms of both context data and remaining times. In particular, the discovery of different trace clusters is carried out by the *Predictive Clustering* submodule which groups traces sharing both similar descriptive and target values. This latter module leverages the CLUS system (DLAI Group, 1998), a predictive clustering framework for inducing PCT models out of propositional data.

In this regard, the *Log-View Generator* submodule acts as a "translator" which converts all log traces into propositional tuples, according to the ARFF format used in CLUS. As explained above, this mapping relies on the explicit representation of both context data and target attributes, derived from the original log. In this process, each trace can be enriched with additional (environmental) context features, such as workload indicators and aggregated time dimensions, in case they are not explicitly stored in the log.

These context-enriched traces are then mapped into a transactional form and delivered to the *Pattern Mining* module. This latter, in its turn, will provide the *Log-View Generator* with a set of frequent patterns, which are to be eventually used as target features for the predictive clustering step. Specifically, the extraction of frequent patterns is carried out by the *Pattern Mining* module in two steps: first patterns are first mined (by the *Pattern Extractor* submodule), and then

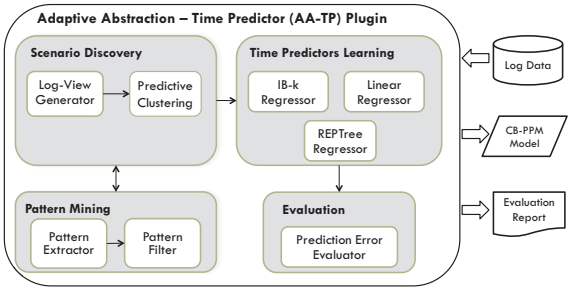


Figure 3: Logical architecture of the AA-TP system.

the most relevant of them are selected (by the *Pattern Filter* submodule).

Once the predictive clustering procedure has been completed, the traces (each labeled with a cluster ID) are delivered to the *Time Predictors Learning* module, which implements a range of classical regression algorithms (including, in particular, IB-k, Linear Regression, and RepTree). These algorithms are eventually used to induce the local predictor (i.e., PPM) of each discovered cluster, which will compose (together with the logical rules discriminating among the clusters) the overall CB-PPM model, returned as the main result. For inspection purposes and further analysis, such a model is stored in a repository. Finally, the *Evaluation* module helps the user assess the quality of time predictions on a generic test set.

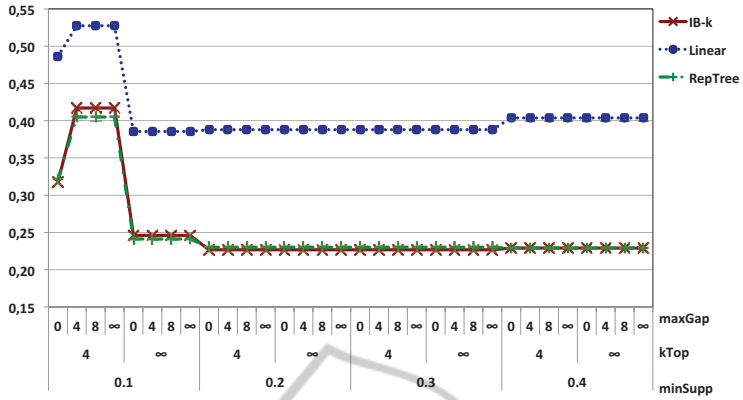
5 EXPERIMENTS

This section illustrates some experimental activities that we conducted, on real data, with the prototype system AA-TP, implementing a specialized version of *AA-PPM Discovery* algorithm, where the target performance measure is the remaining processing time (i.e., the time needed to complete a partial process instance) — as explained in the previous section.

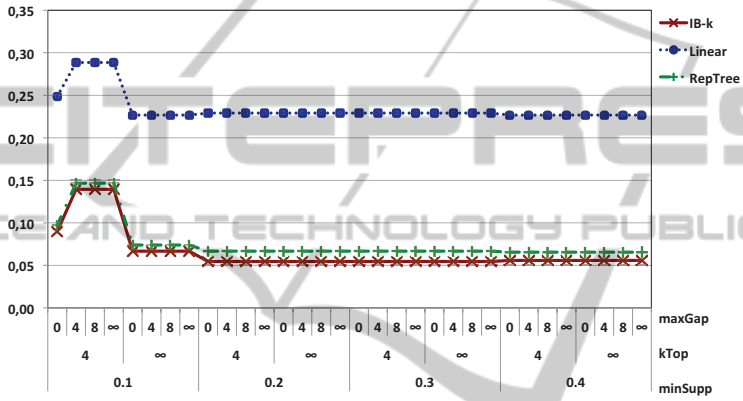
5.1 Testbed

The experiments were performed on the logs of a real transshipment system, mentioned in Example 1 — more precisely, on a sample of 5336 traces, corresponding to all the containers that passed through the system in the first third of year 2006.

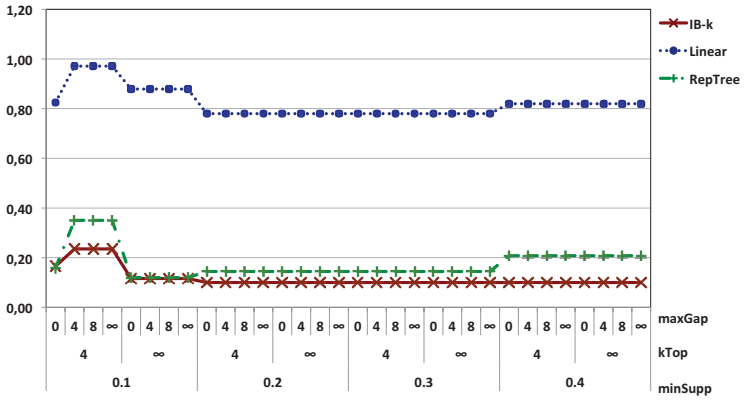
The log stores a series of logistic activities applied to each container passing through a maritime terminal. Basically, each container is unloaded from a ship and temporarily placed near to the dock, until it is carried to some suitable yard slot for being stocked. Symmetrically, at boarding time, the container is first placed in a yard area close to the



(a) The effect of parameters on **rmse** results.



(b) The effect of parameters on **mae** results.



(c) The effect of parameters on **mape** results.

Figure 4: Sensitiveness of AA-TP w.r.t. parameters at varying of kind of regressor.

dock, and then loaded on a cargo. Different kinds of vehicles can be used for moving a container, including, e.g., cranes, “straddle-carriers”, and “multi-trailers”. This basic life cycle may be extended with additional transfers, devoted to make the container approach its final embark point or to leave room for other ones. Several data attributes are available for each container as context data (of the corresponding process instance), including: the origin and final ports, its previous and next calls, various properties of the ship unloading it, physical features (such as, e.g., size, weight), and some information about its contents. Like in (Folino et al., 2012), we also considered a few more (environment-oriented) context features for each container: the hour (resp., day of the week, month) when it arrived, and the total number of containers that were in the port at that moment.

Considering the remaining processing time as the

Table 1: Errors (avg±stdDev) made by AA-TP and its competitors, when using the *bag* abstraction mode.

	AA-TP (IB-k)	AA-TP (RepTree)	CA-TP	FSM
rmse	0.205±0.125	**0.203±0.082**	0.291±0.121	0.505±0.059
mae	**0.064±0.058**	0.073±0.033	0.142±0.071	0.259±0.008
mape	**0.119±0.142**	0.189±0.136	0.704±0.302	0.961±0.040

Table 2: Errors (avg±stdDev) made by AA-TP and its competitors, when using the *set* abstraction mode.

	AA-TP (IB-k)	AA-TP (RepTree)	CA-TP	FSM
rmse	0.287±0.123	**0.286±0.084**	0.750±0.120	0.752±0.037
mae	**0.105±0.061**	0.112±0.035	0.447±0.077	0.475±0.009
mape	**0.227±0.131**	0.267±0.060	2.816±0.303	2.892±0.206

target performance measure, we will measure prediction effectiveness by way of three classic error metrics (computed via 10-fold cross validation): *root mean squared error (rmse)*, *mean absolute error (mae)*, and *mean absolute percentage error (mape)*. For an easier interpretation of results, the former two metrics will be normalized w.r.t. the *average dwell-time (ADT)*, i.e., the average length of stay over all the containers that passed through the terminal. In this way, all the quality metrics will be dimensionless (and hopefully ranging over [0,1]). Moreover, for the sake of statistical significance, all the error results shown in the following have been averaged over 10 trials.

5.2 Test Results: Tuning of Parameters

We tried our approach (referred to as AA-TP hereinafter) with different settings of its parameters, including the base regression method (*REGR*) for inducing the PPM of each discovered cluster. For the sake of simplicity, we here only focus on the usage of two basic regression methods: classic *Linear* regression (Draper and Smith, 1998), and the tree-based regression algorithm *RepTree* (Witten and Frank, 2005). In addition, we consider the case where each PPM model simply encodes a *k*-NN regression procedure (denoted by *IB-k* hereinafter), as a rough term of comparison with the family of instance-based regression methods (including, in particular, the approach in (van Dongen et al., 2008)). For all of the above regression methods, we reused the implementations available in the popular data-mining library Weka (Frank et al., 2005). We remind that the other parameters are: *minSupp*, (i.e., the minimum support for frequent patterns), *kTop* (i.e., the maximal number of patterns to keep, and then use in the clustering), and *maxGap* (i.e., the maximal number of spurious events allowed to occur among those of a given pattern, in a trace that supports this latter). Figure 4 allows for analyzing the three kinds of errors varies, respectively, when using different regression methods (distinct curves are depicted for them), and different values of the parameters (namely, $maxGap = 0,4,8,\infty$, $kTop = 4,\infty$, and $minSupp = 0.1,\ldots,0.4$).

Clearly, the underlying regression method is the factor exhibiting a stronger impact on precision results. In particular, the disadvantage of using linear regression is neat (no matter of the error metrics), whereas both *IB-k* and *RepTree* methods performs quite well, and very similarly. This is good news, especially for the *RepTree* method, which is to be preferred to *IB-k* for scalability reasons. Indeed, this latter may end up being too time-consuming at run-time, when a large set of example traces must be kept – even though, differently from pure instance-based methods (like (van Dongen et al., 2008)), we do not need to search across the whole log, but only within a single cluster (previously selected, based on context data).

As the to remaining parameters, it is easily seen that poorer results are obtained when $minSupp = 0.1$ and $kTop = 4$, as well as when $minSupp = 0.4$. As a matter of fact, the former case epitomizes the cases where we cut little (according to frequency) during the generation of patterns, while trying to reduce their number in the filtering phase; the latter, instead, is an opposite situation where a rather high threshold support threshold is employed, at a higher risk of loosing important pieces of information on process behaviour. In more detail, in the former case, the negative outcome is alleviated when setting $maxGap = 0$, i.e., the patterns are required to exactly match a segment (i.e., subsequence of contiguous elements) in their supporting traces. It is worth noticing that, apart from these extreme cases, AA-TP exhibits good stability and robustness, over a wide range of parameter settings. Remarkably, when *minSupp* gets a value from $[0.2,\ldots,0.3]$, the remaining two parameters, namely *kTop* and *maxGap*, do not seem to affect the quality of predictions at all. In practice, it suffices to choose carefully the regression method (and a middling value of *minSupp*) to ensure good and stable prediction outcomes, no matter of the other parameters – which would be, indeed, quite harder to tune in general.

5.3 Comparison with Competitors

Let us finally compare our approach with two other ones, defined in the literature for the discovery of a PPM: CA-TP (Folino et al., 2012) and FSM (van der Aalst et al., 2011). Tables 1 and 2 reports the average errors (and associated standard deviations) made by system AA-TP, while varying and the base regression method (namely, *Linear*, *RepTree* and *IB-k*). In par-

ticular, the first table regard the case when bag representations are used for abstracting traces, whereas the second corresponds to the usage of set abstractions.

These values were computed by averaging the ones obtained with different settings of the parameters *minSupp*, *kTop*, and *maxGap*. Similarly, for both of the approaches CA-TP and FSM, we computed the average of the results obtained using different values of the history horizon parameter h (precisely, $h = 1, 2, 4, 8, 16$), and the best-performing setting for all the remaining parameters — which are of minor interest here, since we mainly want to contrast our abstraction strategy to the classical ones of competitors.

Interestingly, the figures in Tables 1 and 2 indicate that our approach is more accurate than both competitors, irrespective of the abstraction strategy adopted. It is worth noticing that the best results (shown in bold in the tables), for all the error metrics, are obtained when AA-TP is used with the *bag* abstraction mode. Indeed, when combining this abstraction mode with the *IB-k* regressor, AA-TP manages to lower the prediction error by about 55.9% on average w.r.t. CA-TP, and by an astonishing 74.1% w.r.t. FSM, on average (w.r.t. all the error metrics). Similar results are obtained when using *RepTree* (still with bag abstractions), where a reduction of 50.7% (resp., 70.6%) is achieved w.r.t. to CA-TP (resp., FSM).

6 CONCLUSIONS

We have presented a new predictive process-mining approach, which fully exploits context information, and determines the right level of abstraction on log traces in data-driven way. Combining several data mining and data transformation methods, the approach allows for recognizing different context-dependent process variants, while equipping each of them with a separate regression model.

Encouraging results were obtained on a real application scenario, showing that the method is precise and robust, and it yet requires little human intervention. Indeed, it suffices not to use extreme values for the support threshold to have low prediction errors, no matter of the other parameters (i.e., *maxGap* and *kTop*) — which would, indeed, harder to tune.

As future work, we plan to explore the usage of sequence-like patterns (e.g., k-order subsequences) — possibly combined with those already considered here — in order to capture the structure of a process instance in a more precise (but still abstract enough) manner, as well as to fully integrate our approach into a real Business Process Management platform, in order to offer advantage run-time services.

REFERENCES

Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In *Proc. of 20th Int. Conf. on Very Large Data Bases (VLDB'94)*, pages 487–499.

Blockeel, H. and Raedt, L. D. (1998). Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297.

DLAI Group (1998). CLUS: A predictive clustering system. Available at *http://dtai.cs.kuleuven.be/clus/*.

Draper, N. R. and Smith, H. (1998). *Applied Regression Analysis*. Wiley Series in Probability and Statistics.

Folino, F., Guarascio, M., and Pontieri, L. (2012). Discovering context-aware models for predicting business process performances. In *Proc. of 20th Int. Conf. on Cooperative Information Systems (CoopIS'12)*, pages 287–304.

Frank, E., Hall, M. A., Holmes, G., Kirkby, R., and Pfahringer, B. (2005). Weka - a machine learning workbench for data mining. In *The Data Mining and Knowledge Discovery Handbook*, pages 1305–1314.

Hardle, W. and Mammen, E. (1993). Comparing nonparametric versus parametric regression fits. *The Annals of Statistics*, 21(4):1926–1947.

Harlde, W. (1990). *Applied NonParametric Regression*. Cambridge University Press.

Quinlan, R. J. (1992). Learning with continuous classes. In *In Proc. of 5th Australian Joint Conference on Artificial Intelligence (AI'92)*, pages 343–348.

van der Aalst, W. M. P. and *et al.* (2007). ProM 4.0: Comprehensive support for real process analysis. In *Proc. of 28th Int. Conf. on Applications and Theory of Petri Nets and Other Models of Concurrency (ICATPN'07)*, pages 484–494.

van der Aalst, W. M. P., Schonenberg, M. H., and Song, M. (2011). Time prediction based on process mining. *Information Systems*, 36(2):450–475.

van der Aalst, W. M. P., van Dongen, B. F., Herbst, J., Maruster, L., Schimm, G., and Weijters, A. J. M. M. (2003). Workflow mining: a survey of issues and approaches. *Data & Knowledge Engineering*, 47(2):237–267.

van Dongen, B. F., Crooy, R. A., and van der Aalst, W. M. P. (2008). Cycle time prediction: When will this case finally be finished? In *Proc. of 16th Int. Conf. on Cooperative Information Systems (CoopIS'08)*, pages 319–336.

Witten, I. H. and Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition*. Morgan Kaufmann Publishers Inc.