Context-Aware Prediction on Business Process Executions

Francesco Folino, Massimo Guarascio, and Luigi Pontieri

Institute for High Performance Computing and Networking (ICAR)
National Research Council of Italy (CNR)
Via Pietro Bucci 41C, I87036 Rende (CS), Italy
{ffolino,guarascio,pontieri}@icar.cnr.it

Abstract. Discovering predictive performance models is an emerging topic in Process Mining. However, making accurate estimates is not easy especially when considering fine-grain metrics (such as processing times) on complex and flexible processes, where performances may change over time depending on context factors. We try to face such a situation by a general predictive-clustering approach, where different context-related execution scenarios are find and equipped with distinct performance-prediction models. A two-stage forecast can be then made for a new process case by using the model of the cluster it is estimated to belong to. Tests on real-life logs confirmed the validity of the approach.

1 Introduction

Process mining techniques [6] are a precious tool for analysing business processes, owing to their capability to extract useful information out of historical process logs. While most traditional approaches focused on the discovery of control-flow models, increasing attention has been gained by the discovery of predictive process models, providing operational support at run-time. In particular, an emerging research stream [7, 5] concerns the induction of models for forecasting performances metrics on new process instances.

However, accurate forecasts are not easy to make for fine-grain measures (like, e.g., processing times), especially when the analyzed process exhibits complex and flexible dynamics, and its execution schemes and performances change over time, depending on the context.

In general, more precise process models can be found by exploiting ad-hoc clustering methods [4], while regarding each resulting cluster as evidence for a peculiar execution scenario. However, all previous clustering-oriented process mining approaches focused on control-flow aspects, without spending any effort towards improve performance predictors.

In this paper, which summarizes and generalizes the approach in [3], we describe a general predictive-clustering computation scheme, meant to detect different context-related execution scenarios (or *process variants*), and to equip each of them with a process performance model. The final outcome is a novel kind of

model, which can effectively support performance forecasts on novel (ongoing) process instances.

After introducing some preliminary concepts in Section 2, we describe the general approach, along with a concrete implementation of it, in Section 3, and some experimental findings in Section 4.

2 Preliminaries and Formal Framework

As usual, we assume that for each process instance (a.k.a "case") a *trace* is recorded, encoding the sequence of *events* happened during its enactment. Let $\mathcal{T}$ be a reference universe of all (possibly partial) traces that may appear in a log. For any *trace* $\tau \in \mathcal{T}$, $len(\tau)$ is the number of events recorded in τ; moreover, for $i = 1 \,..\, len(\tau)$, $\tau[i]$ denotes the i-th event of τ, while $task(\tau[i])$ and $time(\tau[i])$ are the task and timestamp associated with $\tau[i]$, respectively. Two tuples are also defined for τ, characterizing its execution context: *(i)* ("intrinsic") *data properties*, denoted by $data(\tau)$, and *(ii)* ("extrinsic") *environmental features*, denoted by $env(\tau)$, and capturing the state of the BPM system when τ started. For short, $context(\tau)$ denotes the juxtaposition of $data(\tau)$ and $env(\tau)$. Moreover, $\tau(i]$ is a *prefix* sub-trace of τ, for $i = 0 \,..\, len(\tau)$, which contains the first i events of τ and the same context data (i.e., $context(\tau(i] = context(\tau))$.

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.

Let $\hat{\mu} : \mathcal{T} \to \mathbb{R}$ be an (unknown) function assigning a performance value to any (possibly unfinished) process trace. For the sake of concreteness, we hereinafter focus 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). We finally assume that such a performance value is known for any prefix trace in $\mathcal{P}(L)$, for any given log L. Indeed, for any log 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, for short) is a model that can predict the performance value (i.e., remaining time) of any process enactment, based on its associated event sequence. Such a model can be viewed as a function $\mu : \mathcal{T} \to \mathbb{R}$ estimating $\hat{\mu}$ all over the trace universe. Learning a PPM is then a special induction problem, where the training set is 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 (e.g., [5, 7]) share the idea of regarding traces at some appropriate level of abstraction – performances hardly depends on the particular sequence of events occurred, but rather on certain properties of (some of) them. Some examples of such trace abstraction functions are defined next.

Definition 1 (Trace Abstraction Function). Let $h \in \mathbb{N}^+ \cup \{\infty\}$ be a threshold on past history. A *trace abstraction function* $abs_h^{mode} : \mathcal{T} \to \mathcal{R}$ is a function mapping each trace $\tau \in \mathcal{T}$ to an element $abs_h^{mode}(\tau)$ in a space $\mathcal{R}$ of abstract representations[1]. For any $\tau \in \mathcal{T}$, let us denote $n{=}len(\tau)$ and $j{=}n{-}h{+}1$ if $n > h$, and

[1] Each $\alpha \in \mathcal{R}$ is a high level representation of some traces, meant to capture some performance-relevant state of the process analyzed.

j=1 otherwise. Then we define: ***(i)*** $abs_h^{list}(\tau) = \langle task(\tau[j]), \ldots, task(\tau[n])\rangle$, ***(ii)*** $abs_h^{bag}(\tau) = [\ (t,p) \mid t \in abs_h^{set}(\tau)$ and $p = |\{\tau[k]|\ j \leq k \leq n$ and $task(\tau[k]) = t\}|\ ]$, and ***(iii)*** $abs_h^{set}(\tau) = \{\ task(\tau[j]), \ldots, task(\tau[n])\ \}$. □

Example 1. We next introduce our running example, inspired to a real-life case study (also used for validating our approach) concerning a transshipment process. 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 the following ones: moving c by means of either a straddle-carrier (MOV) or a shore crane (OUT), and shuffling c with another container (SHF). Several data attributes are available for τ_c, including physical properties (e.g., size, weight) of container c, its previous and next calls (namely `PrevHarbor` and `NextHarbor`), the navigation lines delivering/loading c (namely `NavLine_IN` and `NavLine_OUT`). A few additional features are computed for τ_c to characterize the state of the transshipment system at the time, say t_c, when c arrived at the harbor: the hour (`ArrivalHour`), day of the week (`ArrivalDay`) and month (`ArrivalMonth`) extracted from t_c, and a rough `Workload` indicator (counting how many containers were in the harbor at time t_c). Let τ be a log trace associated with 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 functions in Def. 1, it is easy to see that for the prefix $\tau(2]$ (i.e., the partial enactment consisting of the first two events) it is: $abs_\infty^{list}(\tau(2]) = \langle MOV, MOV\rangle$, $abs_\infty^{bag}(\tau(2]) = [MOV^2]$, $abs_\infty^{set}(\tau(2]) = \{MOV\}$. For short, any pair (t,p) in a bag is here denoted by t^p (or simply t when $p = 1$). With horizon $h = 1$, $abs_1^{list}(\tau(2])$, $abs_1^{bag}(\tau(2])$ and $abs_1^{bag}(\tau(2])$ just consist of element MOV, which is indeed the last task in $\tau(2]$. ◁

The core idea of *Predictive Clustering* approaches [2] is that, based on a suitable clustering model, predictions for new instances can be based on the cluster where they are estimated to belong. Two kinds of features are considered for any element z in the given instance space $Z = X \times Y$: *descriptive* features and *target* ones (to be predicted). Then, a *predictive clustering model (PCM)* is a function $m : X \to Y$ of the form $m(x) = p(c(x), x)$, where $c : X \to \mathbb{N}$ is a partitioning function and $p : \mathbb{N} \times X \to Y$ is a (possibly multi-target) prediction function.

A specific sub-class of such models can be defined for log traces annotated with both context and performance data, as are those considered in our setting.

Definition 2 (Context-Aware Performance Prediction Model (`CA-PPM`)). Given a log L, a *context-aware performance prediction model* (`CA-PPM`) for L is a pair $M = \langle c, \langle \mu_1, \ldots, \mu_k\rangle\rangle$, where k is the number of different clusters found for L. Model M encodes the unknown performance function $\hat{\mu}$ in terms of a predictive clustering model μ^M, such that: *(i)* $c : context(\mathcal{T}) \to \{1, \ldots, k\}$, *(ii)* $\mu_i : \mathcal{T} \to \mathbb{R}$, for $i \in \{1, \ldots, k\}$, and *(iii)* $\mu^M(\tau) = \mu_j(\tau)$ with $j = c(context(\tau))$. □

Our ultimate goal is to find a `CA-PPM` M such that, for any (possibly partial) trace τ, the associated performance value $\hat{\mu}(\tau)$ is well approximated by $\mu^M(\tau)$. Performance predictions will hence rely on a partitioning function c, assigning (possibly novel) process instances to trace clusters (based on their context data), and on multiple process performance predictors μ_i (one for each cluster).

3 Solution Approach

Seeking an explicit encoding for the hidden performance measure $\hat{\mu}$, based on a given log L, can be stated as the search for a `CA-PPM` (cf. Def. 2) minimizing some loss measure, like those in [2]. However, in order to prevent long computations, we follow a heuristics approach, where a `CA-PPM` is discovered by solving two subproblems: *(P1)* find a function c (locally) minimizing the loss on a propositional view of the input log, which summarizes the correlation between execution patterns and performance values; and *(P2)* learn the predictors μ_i out of the clusters generated by c. This approach frees us from the burden of simultaneously searching over each possible log partition and its associated prediction functions, and allows to reuse existing methods for inducing predictive clustering models (for relational data), and (single-cluster) process-performance predictors (e.g., [5, 7]). The reason why we resort to a propositional view of the log is our belief that directly applying classic predictive clustering methods to process logs is likely to yield poor scalability and accuracy results. In particular, we dismiss the idea of using the partial traces in $\mathcal{P}(L)$ as training instances, as this would incur in two drawbacks: (i) a higher number of training samples (especially for logs of complex and flexible processes); and (ii) higher variability of the target function (since the remaining time progressively decreases along each enactment).

In order to build such a propositional view, a set of target features must be defined for each trace τ, which is associated, indeed, with a sequence of time values — namely, $\hat{\mu}(\tau(1]), \hat{\mu}(\tau(2]), \ldots, \hat{\mu}(\tau)$. Heuristically, each trace is mapped into a vector space, whose dimensions coincide with relevant states of the (hidden) process performance model. Such target features are defined by way of the trace abstraction functions in Def. 1. Specifically, given an abstraction function $abs : \mathcal{T} \to \mathcal{R}$, a "candidate" target feature corresponds to each abstract (state) representation $\alpha \in \mathcal{R}$, and the value $val(\tau, \alpha)$ of this feature for any trace τ is $val(\tau, \alpha) = agg(\ \langle\ \hat{\mu}(\tau(i_1]), ..., \hat{\mu}(\tau(i_s])\ \rangle\ )$, where $\{i_1, ..., i_s\} = \{j \in \mathbb{Z} \mid 0 \leq j \leq len(\tau) \text{ and } abs(\tau(i]) = \alpha\}$, and $i_j < i_k$ for any $0 \leq j < k \leq s$, while agg is a function aggregating a sequence of measure values into a single one (e.g., the average, median, first, last in the sequence). In our experiments, the last sequence element was always chosen as such an aggregate.

As the number of state abstractions may be very high, an optimal subset of them should be chosen to prevent the clustering algorithm from going through a high-dimensional and sparse search space. To this end, we use an ad-hoc greedy strategy, where some "pivot" state abstractions are identified, based on some suitable scoring function $\phi : \mathcal{R} \times 2^{\mathcal{T}} \to [0, 1]$, which gives each state abstraction $\alpha \in \mathcal{R}$ a score $\phi(\alpha, L)$ quantifying the confidence in α making a good target feature for finding an effective predictive clustering model for L. More precisely:

Definition 3 (Pivot State Abstractions and Performance Sketch). Let L be a log, $abs : \mathcal{T} \to \mathcal{R}$ be a trace abstraction function, and $\sigma \in [0, 1]$ be a relevance threshold. Then, any $a \in \mathcal{R}$ is a *pivot state abstraction* for L and σ w.r.t. abs, if $\phi(\alpha, L) \geq \sigma$. Moreover, $PA_{abs}(L, \sigma)$ is the set of all pivot state abstractions for L and σ w.r.t. abs. Let us denote $PA_{abs}(L, \sigma) = \{\alpha_{j1}, ..., \alpha_{ju}\}$.

Then a *Performance Sketch* $\mathcal{S}$ of L (w.r.t. *abs* and σ) is a summarized (propositional) view of L such as: *(i)* each trace τ in L corresponds to a distinct data instance z_τ in $\mathcal{S}$, *(ii)* $context(\tau)$ are the descriptive features of z_τ and *(iii)* $val(\tau, \alpha_{j1}), ..., val(\tau, \alpha_{ju})$ are the target features of z_τ. □

A performance sketch of a log L is a propositional view of L, allowing for quickly calculating an effective `CA-PPM` for L. Such a sketch is the input for inducing a preliminary (propositional) predictive clustering model (where the target features are an approximated representation of how remaining times vary along process enactments), prior to refining the prediction function by inducing a more expressive and precise process performance model for each cluster.

The whole solution approach is encoded in an algorithmic form below.

Definition 4 (Meta-algorithm `CA-PPM Discovery`). Given a log L, compute a `CA-PPM` model M for L as follows:

1. Derive $context(\tau)$ for each $\tau \in L$ — by suitably computing $env(\tau)$.
2. Extract a set of pivot state abstractions $\{\alpha_1, \ldots, \alpha_s\} = PA_{abs}(L, \sigma)$, based on given abstraction function *abs* and relevance threshold σ.
3. Build a performance sketch $\mathcal{S}$ of L.
4. Induce a (propositional) Predictive Clustering model T from $\mathcal{S}$. Let c and q be the partitioning and prediction functions of T, resp., and $L[1], \ldots, L[k]$ be the clusters found.
5. For each cluster $L[i]$, with $i = 1..k$, induce a PPM model from $L[i]$ (based *abs*), and use it to encode the respective function μ_i.
6. Return $M = \langle\ c,\ \langle\ \mu_1, \ldots, \mu_k\ \rangle\ \rangle$ □

The above computation scheme is parametric w.r.t. three major kinds of tasks: *(i)* convert a process log into a propositional dataset $PA_{abs}(L, \sigma)$, according to some suitable scoring function ϕ (Step 2), *(ii)* induce a multi-target predictive clustering model from such a dataset (Step 4), and *(iii)* induce a single process-performance model from a trace cluster (Step 5). Various instantiations of this scheme could be devised by suitably implementing these tasks. As a concrete example, we next illustrate a specialization of the above approach, named algorithm `CA-TP` (or simply `CA-TP`) hereinafter, which was recently proposed in [3].

3.1 Algorithm `CA-TP`

In [3], the selection of pivot state features (Step 2 in Def. 4) relies on an ad-hoc scoring function, which is meant to prefer features ensuring a good trade-off between support, correlation with descriptive features (i.e., those guiding the partitioning of log traces) and performance values' variability.

As to Step 4, a *Predictive Clustering Tree (PCT)* [2] is adopted as the specific form of predictive clustering model. Such a model, where the cluster assignment function is encoded by a *decision tree*, is learnt with system CLUS [1], through a recursive partitioning of the training set. Roughly speaking, at each step, a split test is greedily chosen, over one descriptive feature, which (locally) minimizes a loss function, according to (prototype-based) intra-class variances.

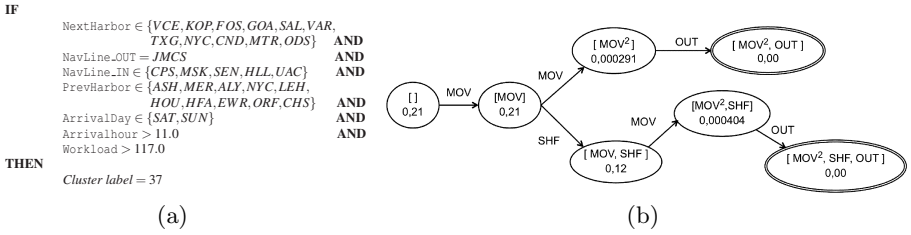

Fig. 1. Excerpt of a `CA-PPM`: (a) decision rule and (b) A-FMS model for one cluster.

With regard to Step 5, the method proposed in [5] is used to discover, for each cluster, a PPM model, in the form of an *Annotated Finite State Machine* (*A-FSM*). Basically, in [5], an FSM is first built, where a one-to-one mapping exists between its nodes and the representations yielded by a given abstraction function abs, while each transition is labelled with an event property (namely, a task label in our setting). Assuming, e.g., that abs^{list}_{∞} is used and a, b, c are three process tasks, a transition labelled with c from state $\langle a, b\rangle$ to state $\langle a, b, c\rangle$ will appear in the resulting FSM model if there is some trace τ in the input log such that $abs^{list}_{\infty}(\tau(i]) = \langle a, b\rangle$ and $abs^{list}_{\infty}(\tau(i+1]) = \langle a, b, c\rangle$. Such a model is eventually turned into an A-FSM, by equipping each node s with some statistics (e.g., the average) computed from the values that $\hat{\mu}$ takes on all trace prefix $\tau \in \mathcal{P}(L)$ such that $abs(\tau)$ coincides with the abstraction of s. The A-FSM model of cluster $L[i]$ is used to implement function μ_i, estimating the remaining time of any (sub-) trace $\tau(j]$ falling in $L[i]$. In case $\tau(j]$ is mapped (by iterated applications of abs) to an unseen sequence of states of the model, the forecast is simply derived from the latest valid estimate computed for the same enactment. Precisely, denoting by $\tau(j']$ the longest prefix of $\tau(j]$ reaching a valid state in the A-FSM model, we estimate $\mu(\tau(j]) = \max\{\ 0,\ \mu_i(\tau(j']) - time(\tau[j]) + time(\tau[j'])\ \}$.

Example 2. Fig. 1 shows an excerpt of a `CA-PPM` which was mined from log data like those described in Example 1, by using algorithm `CA-TP` with the abstraction function abs^{bag}_4. The left side of Fig. 1 reports the decision rule associated with cluster 37, which corresponds to a leaf of the PCT model found with system CLUS [1], and encoding the clustering function c. Notice that cluster membership depends on both case properties and environmental data. On the right side, the A-FSM encoding the prediction function (μ_{37}) of the same cluster is depicted. Notice, by the way, that notation MOV^2, appearing in some of the nodes, means task MOV occurring twice. The performance forecasts produced by this A-FSM for a novel container case clearly depend on the latter 4 tasks it has undergone. In particular, two consecutive MOV tasks are estimated to lead to shorter processing times (w.r.t. the case where the first MOV is followed by a SHF). Let us assume that the following context data are associated with the trace τ of Example 1: `NavLine_IN` $= CPS$, `NavLine_OUT` $= JMCS$, `NextHarbor` $= KOP$, `PrevHarbor` $= ASH$, `Workload`=200, `ArrivalDay` $= SUN$ and `ArrivalHour` $= 16$. Clearly all τ's prefixes are assigned to cluster 37 — in

Table 1. Error reductions (%) of *CA-TP* w.r.t. the baseline method FSM.

Parameters (abs_h^{mode})		**CA-TP**$^-$ (Δ%)			**CA-TP** (Δ%)		
mode	h	*rmse*	*mae*	*mape*	*rmse*	*mae*	*mape*
LIST	1	-0,8%	-1,6%	-0,7%	-1,2%	-1,6%	-5,8%
	2	-28,1%	-51,7%	-27,2%	-28,1%	-55,2%	-31,3%
	4	-26,6%	-50,0%	-41,1%	-65,6%	-71,4%	-72,8%
	8	-25,0%	-50,0%	-57,0%	-64,8%	-71,4%	-73,8%
	16	-25,0%	-50,0%	-57,0%	-64,8%	-71,4%	-73,8%
	Total	$\mathbf{-18,8\%}$	$\mathbf{-33,3\%}$	$\mathbf{-16,8\%}$	$\mathbf{-40,8\%}$	$\mathbf{-44,3\%}$	$\mathbf{-26,2\%}$
BAG	1	-0,8%	-1,6%	-0,7%	-1,2%	-1,6%	-5,8%
	2	-27,7%	-50,0%	-27,8%	-27,7%	-53,8%	-33,0%
	4	-28,1%	-55,2%	-41,0%	-66,4%	-72,4%	-72,0%
	8	-26,6%	-55,2%	-59,6%	-66,4%	-72,4%	-74,5%
	16	-26,6%	-55,2%	-59,6%	-66,4%	-72,4%	-74,5%
	Total	$\mathbf{-19,6\%}$	$\mathbf{-36,0\%}$	$\mathbf{-17,7\%}$	$\mathbf{-41,4\%}$	$\mathbf{-44,9\%}$	$\mathbf{-26,8\%}$
Grand Total		$\mathbf{-19,2\%}$	$\mathbf{-34,7\%}$	$\mathbf{-17,3\%}$	$\mathbf{-41,1\%}$	$\mathbf{-44,6\%}$	$\mathbf{-26,5\%}$

Table 2. Computation times (sec) for method (CA-TP) and its virtual "pseudo-parallel" extension (CA-TP$^+$) and baseline method.

	abs_h^{LIST}			abs_h^{BAG}		
h	**CA-TP**	**CA-TP**$^+$	**FSM**[5]	**CA-TP**	**CA-TP**$^+$	**FSM**[5]
1	16.8	7.4	3.9	17	7.3	4.0
2	20.0	9.7	5.6	19.7	9.6	5.5
4	19.6	7.9	10.7	18.7	7.6	8.4
8	20.2	8.1	16.0	19.8	8.0	10.6
6	92.3	32.6	89.8	79.0	36.0	32.3
Total	**32.0**	**13.1**	**25.2**	**30.9**	**13.7**	**12.2**

actual fact, this happens with no other rule extracted from the discovered PCT model, and the respective forecasts (made by way of the A-FSM in Fig. 1.b) are: $\mu(\tau(0]) = \mu(\tau(1]) = 0.21$, $\mu(\tau(2]) = 0.000291$, and $\mu(\tau(3]) = 0.0$. $\lhd$

4 Experiments

A series of tests are described in this section, which were performed on the logs of the transshipment scenario mentioned in Example 1, using algorithm CA-TP to discover a CA-PPM for the prediction of remaining processing times. Specifically, we report results obtained, with different kinds of abstraction functions, on a sample of 5336 traces, corresponding to all the containers that passed through the system in the first third of year 2006.

Prediction effectiveness was measured, according to a 10-fold cross validation scheme, by way of three classic error metrics: *root mean squared error (rmse)*, *mean absolute error (mae)*, and *mean absolute percentage error (mape)*. All the error results shown in the following were averaged over 10 trials. The associated variances are not reported for lack of space – however, for all metrics, the variance was always lower than 5% of the respective average.

Table 1 summarizes the percentage of error reduction (Δ%) in predicting remaining times obtained by method CA-TP w.r.t. the one proposed in [5] (here denoted by FSM, and also used as a base learner in the former). The tests were performed using different trace abstraction functions abs_h (set-based abstractions are omitted for lack of space), while keeping $\sigma = 0.4$. The results of CA-TP

are further differentiated according to the kinds of context data used as descriptive features: *(i)* CA-TP$^-$ refers to the case where only static container properties (e.g., dimensions, origin/destination ports) are used, whereas *(ii)* CA-TP refers the case where log traces are associated with extrinsic features (namely, workload and seasonality factors), in addition to the former kind of data attributes. Results clearly show that our approach outperforms the baseline one, no matter of parameters' values — Mann-Withney-Wilcoxon test (at 0.05 significance level) confirmed that the methods behave really differently from one another.

Prediction accuracy seems to depend mainly on two factors: the usage of derived context features and the history horizon h. Indeed, the advantage of using environment-related features is neat, as the average error reduction of CA-TP is about 37%, whereas CA-TP$^-$ "only" achieves a 24% reduction. As to h, appreciable benefits are obtained as soon as $h > 1$, with the best performances achieved with $h = 4$ (when each error shrink more than 65% w.r.t. the baseline), while no substantial further improvement is obtained for $h > 8$. The effect of the abstraction mode looks less marked, since very similar (good) results are found in both cases. By the way, poorer performances were obtained with set-oriented trace abstraction functions. However, as our approach confirmed, even in such cases, its superiority to the baseline method, these results omitted.

Table 2 finally shows the average computation times spent by CA-TP (using all context features) and by the baseline, on a dedicated machine with an dual-core Intel processor, 2GB of RAM and Windows XP Pro. The columns named CA-TP$^+$ corresponds an idealistic "overhead-free" virtual parallelization of the method, where the A-FSM models of all clusters are learnt concurrently. As expected, CA-TP always takes longer times than the baseline, yet getting a good trade-off between effectiveness and efficiency. The results of the virtual parallelization CA-TP$^+$ lead us believe in the potentiality of further scalability gains.

References

1. CLUS: A predictive clustering system. Available at *http://dtai.cs.kuleuven.be/clus/*.
2. H. Blockeel and L. De Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, 1998.
3. F. Folino, M. Guarascio, and L. Pontieri. Discovering context-aware models for predicting business process performances. In *Accepted for publication on Proc. of 20th Int. Conf. on Cooperative Information Systems (CoopIS'12)*.
4. G. Greco, A. Guzzo, L. Pontieri, and D. Saccà. Discovering expressive process models by clustering log traces. *IEEE Trans. on Knowl. and Data Engineering*, 18(8):1010–1027, 2006.
5. W. M. P van der Aalst, M. H. Schonenberg, and M. Song. Time prediction based on process mining. *Information Systems*, 36(2):450–475, 2011.
6. W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters. Workflow mining: a survey of issues and approaches. *Data & Knowledge Engineering*, 47(2):237–267, 2003.
7. B. F. van Dongen, R. A. Crooy, and W. M. P. van der Aalst. 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, 2008.