Discovering Context-Aware Models for Predicting Business Process Performances

Seminar - Introduction to Feature Prediction of Running Process Instances

Bianka Bakullari

Process and Data Science Chair
RWTH Aachen University
Germany
bianka.bakullari@rwth-aachen.de

Abstract. Being able to make performance predictions on running process instances is becoming a very attractive research topic in the field of Process Mining, since run-time operational support enables increasing the efficiency of the process on the spot, avoiding missing deadlines, reacting quickly to changes and so on. However, in complex and flexible business processes, the predictions are not always accurate. The approach we briefly describe here attempts at making the prediction models more precise even in very complex situations, by first partitioning the process instances into clusters depending on their contexts and then applying some known prediction models.

1 Introduction

In the field of Process Mining, the main goal is to extract useful information from real event logs in order to better understand and optimize the processes behind the available data. Optimization could imply among other things making processes more efficient, finding bottlenecks and behavioral patterns or making recommendations. One main aspect of Process Mining is control-flow discovery, i.e yielding some model visualising the dependencies between the activities that have happened during the process. In the last years, there have been many attempts to use the available data not only to discover process models that reflect what has already happened, but also to exploit this past information to influence the handling of currently running process instances, e.g., estimate the completion time of some current process instance based on the duration of similar process instances in the past. An intuitive solution adressing this problem is given in [2], where a Finite State Machine (FSM) is constructed. The idea is that each state represents an abstraction of some (partial) trace present in the event data, whereas transitions connect the states in a way that there is a unique path in the state machine for each trace in the event log. Furthermore, the FSM becomes useful for prediction since each state representing some (partial) trace contains some annotation (A-FSM), reflecting the performance value measured over all past traces which at some point had the same state abstraction. The approach we explain in this work also uses A-FSMs based on the same principle, but the attempt to yield useful predictions is taken a step further by first partitioning the process instances into clusters [1]. We assume that the performance of process instances also depends on context features, which can be *internal* (case dependent attributes) or *external* (workload and other given circumstances). The clusters are obtained by using the values of context features to split the set of process instances. After that, for each of the clusters, a corresponding A-FSM is constructed. By looking at a prediction model built based only on cases with similar context, the prediction for some current running process instance is naturally expected to be more precise.

2 Preliminaries

2.1 Notation

Given a function $f : A \to B$ we define $A = domain(f)$ and $B = range(f)$. In this paper we distinguish sets, multisets and sequnces by using the particular brackets $\{\}$, [ ] and $\langle\rangle$ respectively.
Recall that for elements a, b, c we have: $\{a, b, c, b\} = \{a, b, c\} = \{a, c, b\}$, $[a, b, b, c] = [a, b^2, c] = [a, b, c, b] \neq [a, b, c]$ and $\langle a, b, b, c\rangle \neq \langle a, b, c, b\rangle$.

For any sequence s, we use $len(s)$ to describe its length, whereas $s[i]$ for $i \in \{1, ..., len(s)\}$ stands for the ith element of s. Moreover, for a sequence s and $i \in \{1, ..., len(s)\}$, $s[i) := \langle s_1, ..., s_i\rangle$ is the prefix of s of size i.

2.2 Event logs

Let $\mathcal{E}$ be the universe of events, that is, the set of all possible events where each $e \in \mathcal{E}$ is unique. Let $\mathcal{T} = \mathcal{E}^*$ be the universe of sequences of events, which we call *traces*. An event log L is a subset of $\mathcal{T}$. $\mathcal{E}(L)$ describes the set of all events appearing in L, where events are additionally characterized by the following properties: each event corresponds to an $activity\ label$, has a particular $caseID$ and a $timestamp$. These three properties can be viewed as functions mapping each event e appearing in $\mathcal{E}(L)$ to values $label(e), caseID(e)$ and $timestamp(e)$ respectively. Additionally, an event may also contain information on the executor of the event ($resource(e)$), cost of its execution ($cost(e)$) and so on.

Given a log L, for each value $c \in range(caseID)$ the set $\{e \in \mathcal{E}(L) \mid caseID(e) = c\}$ contains all events corresponding to the particular case c in the log. When applying process discovery algorithms on event logs, one is interested on the flow of the cases, so the ordering of the events belonging to a particular case is of crucial importance. This ordering is given by the values of the $timestamp$ property. For $\{e_1, ..., e_n\} \subseteq \mathcal{E}(L)$ belonging to a particular case $c \in range(caseID)$, the trace of case c is some sequence $\sigma = \langle e_{i_1}, ..., e_{i_n}\rangle$ such that for any $i_k, i_l \in \{i_1, ..., i_n\}$ with $i_k < i_l$ where $e_1 = \sigma[i_k]$ and $e_2 = \sigma[i_l]$, it holds that $timestamp(e_1) \leq timestamp(e_2)$. In other words, the $timestamp$ values of the events in a trace are always non-decreasing. In the context of process mining, cases are also called *proces instances*.

When discovering processes, one usually wants to exploit the traces to obtain models which offer some insight into the real processes behind the available data. If all events are unique, then the trace of each case is also unique, so the information gathered in the log is not very useful for further analysis. Process discovery algorithms usually focus on "what" happened during the execution of a case, so the events in the traces are identified through their $label$ function values. These are also called $activities$ or simply $labels$. As the label function may map many events to the same label, many cases may contain the same traces of activities. Recall that by definition, all traces in a log L are unique since they are sequences in $\mathcal{E}^*$. However, when discovering process models behind some log L, one is mostly interested in the multiset of traces containing (possibly) many occurrences of the elements in $\{\langle label(\sigma[1]), ..., label(\sigma[len(\sigma)])\rangle \mid \sigma \in L\}$.

From now on, whenever we use the term trace for some $\sigma \in L$, we refer to its sequence of event labels.

3 Annotated Finite State Machines

3.1 Transition systems

In this section we explain how transition systems can be used to model the processes behind event logs, based on the definitions given in [2].

For some case c and corresponding trace σ with $len(\sigma) = n$, we say σ is a *fully unfolded* trace and for any $i \in \{1, ..., n-1\}$ we call $\sigma[i]$ a *partial* trace. For a given log L, by $\mathcal{P}(L) := \{\sigma[i] \mid \sigma \in L, 1 \leq i \leq len(\sigma)\}$ we define the set of all partial traces in L.

A transition system is a 5-tuple $(S, \Sigma, T, s_{start}, S_{finish})$ where S is the set of states, Σ is the set of action labels and $T \subseteq S \times \Sigma \times S$ the set of transitions. The transition system has some unique initial state $s_{start} \in S$ and a set $S_{finish} \subseteq S$ of finite states. Each possible transition is given by a triple $(s_1, a, s_2) \in T$ which says that the system can go from state s_1 to state s_2 through action a. The set of paths from the initial state to any final state yields the set of valid walks through the system. These valid walks represent the possible behaviour modeled by the system and are given through the sequences of actions corresponding to them.

A transition system that characterizes the observed behaviour given in a log is a model where the valid walks correspond to traces in the log. So for each trace in the log, there should be a corresponding valid path. For a trace having a corresponding path, we say that it is *replayable* in the model.

In the following we explain how S, Σ and T are chosen. The idea is that during runtime, each case is in some particular current state and has a particular (partial) trace corresponding to it. The next state then should depend on the current state and is determined by its history and the next activity.

For a given log L, one could easily construct a transition system by letting $S = \mathcal{P}(L)$, $\Sigma = range(label)$ and $T = \{(\sigma[i-1), \sigma[i], \sigma[i]) \mid \sigma \in L, 1 \leq i \leq len(\sigma)\}$. The unique initial state would be $s_{start} = \langle\rangle$ and $S_{finish} = \{\sigma \mid \sigma \in L\}$. This way each trace in L is replayable in the transition system.

However, when constructing transition systems to represent processes behind event logs, we want the models to represent the data well. By choosing to have a corresponding state in the system for every partial trace in the log, each trace σ has a unique valid path from initial state $\langle\rangle$ to final state σ. If the event log contains very few traces, such model would be very sensitive to all patterns that appear in those few traces, including those traces that appear very rarely. The chance for a new trace being recorded to not be replayable in the model would be rather high. In this case, defining the states to be the set of all partial traces in the log would produce an *overfitting* model. This could be avoided by adjusting the degree of abstraction, that is, by having a model where each state possibly represents more than one unique partial trace from the log. Moreover, different functions can be defined that map the set of partial traces to the set of states, where all offer a different degree of abstraction, depending on the questions at hand or the volume of data available. In the following we will see some examples of these functions.

3.2 Abstraction functions

In certain scenarios, when replaying the traces in the model, one might not want to distinguish traces which only differ on the order of their event labels, or the number of occurrences of some certain labels. In other cases, one may want to distinguish traces based only on the last k events for some number k, or on whether some activity already took place or not. To achieve this, the function which maps the partial traces to the set of states in the transition system is chosen accordingly. This function is also called an *abstraction function*.

An *abstraction function* is some function $abs : \mathcal{T} \rightarrow \mathcal{R}$ which computes a so-called *state representation* for each sequence in $\mathcal{T}$. The set of unique values in $\mathcal{R}$ onto which elements from $\mathcal{P}(L)$ are mapped onto defines the set of states in the transition system.

In the following we give some examples of possible abstraction functions.

Abstraction: Maximal horizon

In this case, only some part of the trace prefix is relevant. For any $h \in \mathbb{N} \cup \{\infty\}$ an abstraction function abs_h can be defined, such that for any sequence $\sigma \in \mathcal{T}$ we have $abs_h(\sigma) = \langle \sigma[j], ..., \sigma[len(\sigma)] \rangle$ where $j = n - h + 1$ for $h < len(\sigma)$ and 1 otherwise. For example, $abs_3(\langle A, B, B, C, D \rangle) = \langle B, C, D \rangle$. For any sequence $\sigma \in \mathcal{T}$ we have $abs_\infty(\sigma) = \sigma$. So by setting $h = \infty$, the whole sequence is considered.

Abstraction: Filtering

In the context of transition systems, filtering is used when the occurrence of particular events should not be relevant in determining the walk of traces in the model. For a set $\mathcal{F}$, an abstraction function $abs_\mathcal{F}$ can be defined such that for any sequence $\sigma \in \mathcal{T}$, $abs_\mathcal{F}(\sigma)$ is the sequence obtained from σ after having removed the elements which are not contained in the set $\mathcal{F}$. For example, $abs_{\{A,B,E\}}(\langle A, C, B, B, D, E \rangle) = \langle A, B, B, E \rangle$.

For a given log L, we may have $\mathcal{F} \subseteq range(label)$. Note that the outfiltered labels are still relevant when replaying the traces in the transition system, they only do not cause a switch from one state to another.

Events can also be filtered based on other properties, e.g. their $resource$ value, for the case where one only wants to model a system for the traces where certain employees are involved.

Abstraction: Sequence, bag or set

These three variants of the abstraction functions determine whether the order or the number of occurrences of the elements in the sequence should be considered.

The sequence abstraction mode is given by the function abs^{list} where for any sequence σ we have $abs^{list}(\sigma) = \sigma$. Given an event log L, abs^{list} maps each element of $\mathcal{P}(L)$ to itself.

The abstraction function abs^{bag} computes the multiset of elements appearing in a sequence. For example $abs^{bag}(\langle A, B, C, B \rangle) = [A, B^2, C]$. Here the ordering of the elements is not important, only the number of their occurrences is.

For the case where one is only interested on whether some activity has taken place or not, the set abstraction mode is the right choice. For any sequence $\sigma \in \mathcal{T}$, the function abs^{set} maps σ to the set of elements appearing in the sequence. For example is $abs^{set}(\langle A, B, C, B \rangle) = \{A, B, C\}$. Here both the order and the number of occurrences of each element are not important.

The functions given above have different degrees of abstraction. An abstraction function is coarser the more sequences it maps to the same state representations. For example, the abs^{set} abstraction is coarser than abs^{bag}, which itself is coarser than abs^{list}. Also, for any j, k with $j < k$, then abs_j is coarser than abs_k.

If important patterns in the behaviour behind the data are ignored by choosing a very coarse abstraction, then valuable information is lost and we say that the model produced is *underfitting*. In this case, each trace present in the log may still be replayable, however, the model allows for many valid walks which do not correspond to any observed traces.

The challenge in building good models is to find a state representation function which is not overfitting or underfitting.

Note that the abstraction functions presented above make up the building blocks of other more complex abstraction functions. Combining them together, one can more precisely define what is to be considered relevant when distinguishing traces between one another. For example, if one wants to distinguish the traces based on the last two activities that have taken place, this could be realized by first applying the abs_2 abstraction and then the abs^{set} abstraction.

Furthermore, given some $mode \in \{list, bag, set\}$ and a $h \in \mathbb{N} \cup \{\infty\}$, the function abs_h^{mode} computes the correponding $mode$- abstraction after having extracted only the last h elements from the sequence.

It is also worth mentioning that one can use abstraction functions to influence not only the set of states, but also the set of actions. Until now we always defined Σ to be the set of event labels, so that the walk of each trace in the transition system is determined by the sequence of its activities. However, one can define abstractions which map singular events to some representation value from which Σ is determined. The events can be mapped to be any property that can be obtained from the information in the data, e.g. each event can be mapped to its $resource$ value in case we want to track the handover of work of running cases in a particular process.

In the following section, we will see how transition systems can be annotated so that they can be used for prediction purposes.

3.3 Performance Prediction Models

Annotated Finite State Models (short: A-FSMs) for prediction purposes were first presented in [2], where for each running case, the remaining time until its completion was predicted based on the information collected from other completed cases. Given a trace $\sigma \in L$, the *remaining time until completion* for any partial trace of σ is given by the following function:

$$RT(\sigma[i)) = timestamp(\sigma[len(\sigma)]) - timestamp(\sigma[i])$$

Note that the remaining time can be computed only for a completed case. When a case is still running, one may be interested in knowing how long it will take until the case is completed. This is especially important if in a process one wants to predict whether some deadline will to be met otherwise a fine has to be payed.

Let us suppose that the log L contains the completed cases that have been recorded and a corresponding transition system has already been constructed after having picked a particular abstraction function abs for the set S of states. Let σ be a partial trace of some case that is still running and we are interested in predicting the remaining time until the case is finished. Obviously, there is some state s onto which the partial trace σ is mapped onto from the abs function. It is reasonable to try to predict the remaining time for the current running case by looking at the remaining times of all cases that visited state s.

For this reason, each state in the already generated transition system is annotated with some value of what we are interested to predict for future running cases.

In the case when the RT value has to be predicted, one can annotate each state $s \in S$ with the multiset $[RT(\sigma_1), ..., RT(\sigma_k)]$ where $\{\sigma_1, ..., \sigma_k\} = abs^{-1}(s)$ is the set of all partial traces of L whose abstraction is s. The RT function is also called a *measurement function*. Measurement functions compute the value of what is aimed to be predicted for any prefix of a full trace from the log.

After annotating each state with a multiset of values from the range of the measurement function, a *prediction* is computed which aggregates a single value from the multiset. In our example, it would be reasonable to choose the average as the prediction function. This way, each state is annotated with a single value reflecting the expected remaining time of all the cases that visited that state.

Note that a good prediction for a future running instance is only possible if there is a state corresponding to the abstraction of its trace and there has been enough data gathered to compute the annotation of the state.

The approach remains the same for any measurement function that is chosen. Instead of looking at the remaining time, one could measure the cost related to the execution of each trace in order to be able to make some cost evaluation for future instances. In other scenarios, one may be interested in knowing whether some activity has taken place, e.g. whether a fine was charged because some particular regulation was violated. For each of the observed complete traces, if the trace contains the activity, one could add a 1 to all states

visited by replaying the trace in the transition system and a 0 otherwise. The prediction for a running case would then reflect the probability that the particular activity takes place in the future. Similar to this, one could predict whether a particular resource will be included in the execution of the case, and so on.

Until now we saw how for any given event log, a single annotated model can be generated and later used for prediction. In very complex processes, cases with similar corresponding traces or walks through the annotated model can still have very different values of the measurement function. This might be due to the distinct circumstances under which those cases are executed. This variance in the measured values of similar traces may yield unprecise predictions for future running cases. Obviously, partitioning the traces so that the variance of the predicted values in each group is low should produce more precise predictive models. The approach described in [1] which we explain in Section 4 assumes that the imprecision in the predictions is caused by the different contexts in which the cases are executed and by generating a separate A-FSM for each group of cases sharing a similar context, the predictions are going to be more precise.

4 Approach: Context-Aware Models

In this section we explain the approach first presented in [1] where Context-Aware Performance Prediction Models are constructed with the aim of producing more precise predictions under the core assumption that context factors influence process performances.

4.1 Context features

In a given log L, we assume that each process instance has additional context features which can be divided into: *"intrinsic"* features or *case attributes* and *"extrinsic"* features or *environmental features*. The case attributes capture the internal properties of some certain case, e.g. amount of credit being applied for, whereas the environmental features capture the external factors under which the case is executed, e.g. workload, resources available and so on.

Let $A_1, ..., A_q$ with associated domains $D^{A_1}, ..., D^{A_q}$ be the case attributes and $B_1, ..., B_r$ with associated domains $D^{B_1}, ..., D^{B_r}$ the environmental features characterizing the cases in some log L. Then for any (partial) trace in L, its *context* is given by $context : \mathcal{P}(L) \rightarrow D^{A_1} \times ... \times D^{A_q} \times D^{B_1} \times ... \times D^{B_r}$.

Note that since the context is a case property, any full trace in L shares the same context with all its prefix traces. More precisely, for any full trace $\sigma \in L$, it holds that $context(\sigma[i)) = context(\sigma)$ for all $i \in \{1, ..., len(\sigma)\}$.

4.2 Problem description

Given a log $L \subseteq \mathcal{T}$, we are interested in computing a prediction function $\mu : \mathcal{T} \rightarrow \mathcal{M}$ where $\mathcal{M}$ contains the range of the predicted values. This way, for any partial trace of some running process instance, the function μ would yield the expected value of that certain process aspect we are interested in predicting. Assuming that L only contains completed cases, the value of the prediction function is already known for all sequences in $\mathcal{P}(L)$. We define $\hat{\mu} : \mathcal{P}(L) \rightarrow \mathcal{M}$ to be the function that yields the predicted values of all partial traces in the log. Note that $\mu \downarrow_{\mathcal{P}(L)} = \hat{\mu}$, that is, the value of μ for sequences in $\mathcal{P}(L)$ is given by $\hat{\mu}$. Note that $\hat{\mu}$ is some measure function like the RT function we handled in Section 3.

We saw how the prediction function can be encoded in the annotations of the A-FSM generated for the log. Now we attempt to first partition the process instances into clusters, such that the membership in each

cluster is determined by the values of the context features. After that, for each cluster, a separated A-FSM is constructed based on the traces present in that cluster.

In [1] this attempt is realized by constructing a Context-Aware Performance Predition Model (short CA-PPM).

Given a log L with context values $range(context(L))$ and known predictions $\hat{\mu} : \mathcal{P}(L) \rightarrow \mathcal{M}$ for all traces in $\mathcal{P}(L)$, a CA-PPM is a pair $(c, \langle \mu_1, ..., \mu_k \rangle)$ encoding a predictive clustering model g_M for $\hat{\mu}$, such that:
(i) $c : context(L) \rightarrow \{1, ..., k\}$,
(ii) $\mu_i : \mathcal{T} \rightarrow \mathcal{M}$ for $i \in c(context(L))$, and
(iii) $g_M(\sigma) = \mu_j(\sigma)$ for $context(\sigma) = j$.
In other words, when given some running process instance, the cluster it should belong to is computed and then the respective prediction function is applied on that instance.

4.3 Predictive Clustering

In *predictive clustering* one attempts to partition the set of instances in such a way that the instances within one cluster are rather similar to one another, whereas elements across different clusters are rather dissimilar. The clustering function takes the form of a decision tree, where the interior nodes contain the conditions under which the instances are partitioned and the leaves contain the elements of each cluster. The corresponding cluster of each new instance is then calculated by following the right path from the root to some leaf. To measure the similarity within one cluster, one could compute the centroid of that cluster (e.g. the mean value of all elements of that cluster) and then calculate the average distance between all elements and the centroid. More precisely, for a given cluster c of size $|c|$ with centroid z and distance measure $dist$, the lower the value of the term $\frac{1}{|c|} \sum_{e \in c} dist(e, z)$, the more similar the elements in that cluster. If the elements are represented by numerical values, one could use the euclidian distance to measure the distances.

One can assess the quality of a clustering function by calculating the weighted average variance in all clusters. More precisely, by calculating $\sum_{c \in C} \frac{|c|}{N} \times \sum_{e \in c} dist(e, z)$ where N is the number of elements and C the set of clusters.

In [1] the so-called Predictive Clustering Trees (short PCTs) are used to partition the process instances into clusters. In the interior nodes, the discriminating conditions take the form of claims given in propositional logic over the values of $context$. That is, in each node, a certain claim is made regarding particular context features. Then the instances are partitioned into those for which the claim is true and those for which the claim is false.

The goal is to apply the right claims in the right order so that the clustering function determined by them induces clusters where the instances within cluster are similar and those across clusters are dissimilar. Obviously, one can find a function that partitions the set of observed process instances perfectly. However, such function would probably assign new process instances to clusters containing process instances very different from them. Again, a good balance between overfitting and underfitting is required.

It remains to explain how the process instances are represented in the calculation of the clustering function. Recall that the incentive to split the process instances into clusters according to their context features came from the fact that constructing one singular A-FSM for all instances would result in imprecise predictions. This is due to the fact that similar traces which visit common states may have very dissimilar prediction values. Therefore, we want to consider the corresponding traces of two process instances as similar if given some abstraction function for the state representation, the set of states their prefixes are mapped to share many common elements and the prediction values they assign to these states are similar. In the next section, we explain how each process instance is represented as a vector which captures these characteristics.

4.4 Solution Approach

Given some log L, for any completed process instance with corresponding trace $\sigma \in L$, the values $\{\hat{\mu}(\sigma[i) \mid i \in \{1, ..., len(\sigma)\}\}$ are known. Let abs be the state abstraction function describing what differences between the traces are to be considered relevant. To compute the clustering, one could also map each fully unfolded trace σ to the vector $\langle \hat{\mu}(\sigma[i)), ..., \hat{\mu}(\sigma)\rangle$. However, given a certain abstraction function for the determination of states, this representation could mistakenly consider many traces to be dissimilar.

Let $S := range(abs)$ be the set of all states onto which the prefix traces of L are mapped by the given abs function. For each $\alpha \in S$ and each $\sigma \in L$ we try to capture the relation between α and σ by computing the following function:

$$val(\alpha, \sigma) = \begin{cases} \text{NULL} & \text{if } abs(\sigma(i]) \neq \alpha \ \ \forall i \in \{0, ..., len(\sigma)\}; \\ agg\big(\{\hat{\mu}_1(\sigma(i_1]), ..., \hat{\mu}_1(\sigma(i_s])\}\big) & \text{otherwise,} \end{cases}$$

where $\{i_1, ..., i_s\} = \{j \in \mathbb{Z} \mid 0 \leq j \leq len(\sigma) \text{ and } abs(\tau(i]) = \alpha\}$.

That is, for each state and for each trace in the log, the val function yields NULL if no prefix of the trace is mapped onto this state. Otherwise, the predicted values of all prefixes which abs maps onto this state are considered and a singular value is aggregated from them (e.g. average, sum, variance, etc.) For $S = \{\alpha_1, ..., \alpha_n\}$, each fully unfolded trace $\sigma \in L$ is mapped to the n-dimensional vector $\langle val(\alpha_1, \sigma), ..., val(\alpha_n, \sigma)\rangle$.

Additionally, a subset of S can be picked so that the (dis)similarity between process instances in the clusters is computed based only the most meaningful states. Given a log L and an abstraction function $abs : \mathcal{P}(L) \rightarrow S$, the relevance of some state $\alpha \in S$ w.r.t the log L is measured by a so-called *scoring function* $\Phi : S \times 2^{\mathcal{T}} \rightarrow [0, 1]$. Then, by picking some threshold $\delta \in [0, 1]$, only those $\alpha \in S$ are considered relevant for which $\Phi(\alpha, L) \geq \delta$. The Φ-values for each $\alpha \in S$ are calculated as follows:

$$\Phi(\alpha, L) = \sqrt[3]{\Phi_{var}(\alpha, L) \times \Phi_{corr}(\alpha, L) \times \Phi_{supp}(\alpha, L)}. \tag{1}$$

Basically, $\Phi_{var}(\alpha, L)$ computes the variability in the set $\{val(\alpha, \sigma) \mid \sigma \in L\}$. The higher the variability, the more different the prediction values of the partial traces that get mapped to this state. Such state helps find clusters showing quite different prediction values. Function $\Phi_{corr}(\alpha, L)$ measures the maximal correlation between the value taken by the val over each trace and the corresponding $context$ of that trace. Such state is one that guides the partitioning of the traces into clusters.
Finally, $\Phi_{supp}(\alpha, L) = 2 \times min(0,5, |\{\sigma \in L \mid val(\alpha, \sigma) > 0)\}|/|L|)$. States which have very low support are not very helpful in discriminating groups of traces.

Given a log L, abstraction function abs and a threshold $\delta \in [0, 1]$, we use $PA_\delta(L, abs)$ to describe the set of relevant states (also called pivot state abstractions). For $PA_\delta(L, abs) = \{\alpha_1, ..., \alpha_s\}$, each trace $\sigma \in L$ is represented as the vector $\langle val(\alpha_1, \sigma), ..., val(\alpha_s, \sigma)\rangle$ of dimension s.

After having computed the clusters for the given vector representations of the traces in the log, a A-FSM is constructed for each cluster as described in Section 3.

Whenever we want to obtain a prediction for a new running case, we use its $context$ to determine the cluster the case belongs to and then replay its corresponding partial trace in the A-FSM generated for that particular cluster. The annotation of the state that is visited last is the value of the prediction.

Algorithm 1 is the one given in [1] describing the approach.

Input: A log L over some trace universe $\mathcal{T}$, with associated data attributes $A = A_1, \ldots, A_q$ and environment features $B = B_1, \ldots, B_r$, a target measure $\hat{\mu}$ known over $\mathcal{P}(L)$, a trace abstraction function abs, and a relevance threshold $\delta \in [0, 1]$.
Output: A `CA-PPM` model for L (fully encoding $\hat{\mu}$ all over $\mathcal{T}$)

1 Associate a vector $context(\sigma)$ with each trace $\tau \in L$
2 Compute a set $PA_\sigma(L, abs)$ of pivot state abstractions
3 Let $PA_\sigma(L, abs) = \{\alpha_1, \ldots, \alpha_s\}$
4 Build a *performance sketch* $\mathcal{S}$ for L using both context vectors and $PA_\sigma(L, abs)$
// $\mathcal{S} = \Big\{ \big(id(\sigma),\ \ context(\sigma),\ \ \langle val(\alpha_1, \sigma), \ldots, val(\alpha_s, \sigma)\rangle\big) \ \mid\ \sigma \in L \Big\}$
5 Compute a PCT T, with classification (resp. prediction) function c (resp. q), by using $context(\sigma)$ (resp. $val(\alpha_i, \sigma)$, $i = 1 \ldots s$) as descriptive (resp. target) features $\forall \sigma \in L$
6 Let $L[1], \ldots, L[k]$ denote discovered clusters, with $\{1, \ldots, k\} = c(\mathcal{S})$
7 **foreach** $L[i]$ **do**
8 Induce an FSM model f from $L[i]$, using abs as abstraction function
9 Derive an A-FSM f^+ model from f
10 Define prediction function $\mu_i : \mathcal{T} \to \mathcal{M}$ (for cluster i) based on f^+
11 **end**
12 **return** $\langle c, \{\mu_1, \ldots, \mu_k\}\rangle$

Algorithm 1: `CA-PPM Discovery`

5 Example of application

In this section, we apply the approach we described above on the small event log L containing fictional data (see Table 1).

CaseID	Trace	PK	Field	Supervisor	Final Grade
0	[I, M, M, F, M, S]	average	SE	John	0.700
1	[I, M, M, F, R, S]	high	DS	John	0.850
2	[I, M, M, F, R, S]	high	DS	John	0.950
3	[I, M, M, R, F, S]	high	DS	Brown	0.900
4	[I, M, R, F, S]	average	SE	Brown	0.800
5	[I, M, R, F, S]	high	SE	Brown	0.750
6	[I, M, M, F, M, S]	average	DS	John	0.700
7	[I, M, R, M, S]	high	SE	John	0.700
8	[I, M, F, M, M, S]	average	DS	John	0.600
9	[I, M, F, R, S]	high	SE	Brown	0.800

Table 1. Event log with ten traces representing the process of preparation and supervision of an academic research paper

Consider the scenario where in some university, the Computer Science students must deliver a small first academic paper with the intention of introducing them to the basics of scientific research. Here we assume that the students can pick the field of their topic (data attribute *Field*), namely either Data Science (DS) or Software Engineering (SE) and each of them is supervised either by Mr. Jones or by Mr. Brown (data attribute *Supervisor*), a decision made by the supervisors. We suppose that students have different levels of prior knowledge in their topics which may influence their final grade (see *Final Grade*) which has range $[0, 1]$. To capture this, we use the data attribute Prior Knowledge (*PK*) which takes the values *high* or *average*. Each student has 100 days in disposal to submit the completed version of their paper (denoted with label S) and hold a talk presenting their work, but they are rewarded if they deliver their work before the deadline. Each of them is acquired to take part in the introductory meeting (event label I) and they have the chance to once submit some non-final version of their work to their supervisor in order to get some concrete feedback. In the data this is denoted with the label F. During the preparation time, each student can also arrange meetings with his or her supervisor (event label M) while they also have the possibility of rehearsing their presentation (label R). To keep the example intuitive, each timestamp represents a day from 1 to 100 and the introductory meeting takes place on day 1, while the work has to have been submitted until day 100. Otherwise, there are penalties regarding the grade which we choose to ignore here. All activities M, F and P are not mandatory for the course and are only offered to help the students succeed in their task.

For each (partial) trace $\tau(i]$, we can observe the remaining time (*RT*) until submission for case $caseID(\tau)$ which is calculated as follows:

$$RT(\tau(i]) = timestamp(S) - timestamp(\tau[i]).$$

Now suppose that in the future, depending on what state of preparation they are in, the students want to find out if they can finish the task on time and what the final grade would possibly be. Since the meetings, the non-final submission and the presentation rehearsal are optional, they may be interested in finding the best approach to possibly improve their schedule by looking at the results of the students who had the same background as them, that is, same prior knowledge, project field and supervisor. So to define the "state of preparation", we want to look at the number of meetings (M) and the occurrences of F and P along the whole process. Therefore, here we choose to use the abs^{bag}_{∞} function to map each (partial) trace to the multiset of its activities.

The example log in Table 1 contains ten traces which we denote with τ_i for $i \in CaseID$.

Formally, we have $A_1 = PK \in \{average, high\}$, $A_2 = Field \in \{DS, SE\}$ and $B_1 = Supervisor \in \{Jones, Brown\}$. For each (partial) trace $\tau \in \mathcal{P}(L)$, depending on its context: $context(\tau) \in A_1 \times A_2 \times B_1$, we want to estimate the function $\hat{\mu} : \mathcal{P}(L) \to \mathcal{M} = \mathbb{N} \times [0, 1]$, such that:

$$\hat{\mu}(\tau) = \Big(avg(\{RT(\tau')|\tau' \in \mathcal{P}(L), \tau' = \tau\}), avg(\{final\_grade(\tau')|\tau' \in \mathcal{P}(L), \tau' = \tau\})\Big)$$

where avg stands for the average function.

So for each partial trace in the given log, $\hat{\mu}$ estimates the average remaining time until submission and the average grade over the RT and $final\_grade$ values of all equal (partial) traces seen in the log. The computed values can be obtained from Table 2.

We use $\hat{\mu}_i$ for $i \in \{1, 2\}$ to refer to the ith entry of the value of $\hat{\mu}$.

We assume that the value of the final grade is only known after the paper has been submitted, so it is subject to prediction when the case is still running.

For each of the 23 prefixes that are present in the data, there is a corresponding prediction value, state abstraction and a list of cases related to it. Overall there are 15 distinct states and each of them has a

Prefix	Prediction for prefix	CaseIDs containing prefix	Corresponding state	CaseIDs containing state	Annotation for state
[ ]	(99.5, 0.775)	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]	{}	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]	(99.5, 0.775)
[I]	(98.5, 0.775)	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]	{(I, 1)}	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]	(98.5, 0.775)
[I, M]	(60.0, 0.775)	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]	{(I, 1), (M, 1)}	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]	(60.0, 0.775)
[I, M, M]	(40.0, 0.82)	[0, 1, 2, 3, 6]	{(I, 1), (M, 2)}	[0, 1, 2, 3, 6]	(40.0, 0.82)
[I, M, M, F]	(16.25, 0.8)	[0, 1, 2, 6]	{(F, 1), (I, 1), (M, 2)}	[0, 1, 2, 6, 8]	(18.125, 0.7)
[I, M, M, F, M]	(7.5, 0.7)	[0, 6]	{(M, 3), (I, 1), (F, 1)}	[0, 6, 8]	(11.25, 0.65)
[I, M, M, F, M, S]	(0.0, 0.7)	[0, 6]	{(M, 3), (I, 1), (F, 1), (S, 1)}	[0, 6, 8]	(0.0, 0.65)
[I, M, M, F, R]	(5.0, 0.90)	[1, 2]	{(F, 1), (I, 1), (M, 2), (R, 1)}	[1, 2, 3]	(7.5, 0.90)
[I, M, M, F, R, S]	(0.0, 0.90)	[1, 2]	{(F, 1), (M, 2), (S, 1), (I, 1), (R, 1)}	[1, 2, 3]	(0.0, 0.90)
[I, M, M, R]	(20.0, 0.9)	[3]	{(I, 1), (M, 2), (R, 1)}	[3, 7]	(15.0, 0.8)
[I, M, M, R, F]	(10.0, 0.9)	[3]	{(F, 1), (I, 1), (M, 2), (R, 1)}	[1, 2, 3]	(7.5, 0.90)
[I, M, M, R, F, S]	(0.0, 0.9)	[3]	{(F, 1), (M, 2), (S, 1), (I, 1), (R, 1)}	[1, 2, 3]	(0.0, 0.90)
[I, M, R]	(38.33, 0.75)	[4, 5, 7]	{(I, 1), (M, 1), (R, 1)}	[4, 5, 7]	(38.33, 0.75)
[I, M, R, F]	(15.0, 0.775)	[4, 5]	{(F, 1), (I, 1), (M, 1), (R, 1)}	[4, 5, 9]	(15.0, 0.79)
[I, M, R, F, S]	(0.0, 0.775)	[4, 5]	{(M, 1), (F, 1), (S, 1), (I, 1), (R, 1)}	[4, 5, 9]	(0.0, 0.79)
[I, M, R, M]	(10.0, 0.7)	[7]	{(I, 1), (M, 2), (R, 1)}	[3, 7]	(15.0, 0.8)
[I, M, R, M, S]	(0.0, 0.7)	[7]	{(I, 1), (S, 1), (M, 2), (R, 1)}	[7]	(0.0, 0.7)
[I, M, F]	(37.5, 0.7)	[8, 9]	{(F, 1), (I, 1), (M, 1)}	[8, 9]	(37.5, 0.7)
[I, M, F, M]	(20.0, 0.6)	[8]	{(F, 1), (I, 1), (M, 2)}	[0, 1, 2, 6, 8]	(18.125, 0.7)
[I, M, F, M, M]	(15.0, 0.6)	[8]	{(M, 3), (I, 1), (F, 1)}	[0, 6, 8]	(11.25, 0.65)
[I, M, F, M, M, S]	(0.0, 0.6)	[8]	{(M, 3), (I, 1), (F, 1), (S, 1)}	[0, 6, 8]	(0.0, 0.65)
[I, M, F, R]	(15.0, 0.8)	[9]	{(F, 1), (I, 1), (M, 1), (R, 1)}	[4, 5, 9]	(15.0, 0.79)
[I, M, F, R, S]	(0.0, 0.8)	[9]	{(M, 1), (F, 1), (S, 1), (I, 1), (R, 1)}	[4, 5, 9]	(0.0, 0.79)

Table 2. Predictive function $\hat{\mu}$ evaluated over $\mathcal{P}(L)$ together with the corresponding set of states, their annotations and cases related to them

corresponding annotation which can all be seen in Table 2. Note that each state is represented as a set containing pairs (A, x), where A is the activity and x is the number of occurrences of that activity.

To keep the example simple, we dismiss the selection of the most relevant states by taking the set of all states $S := \{abs^{bag}_{\infty}(\tau) \mid \tau \in \mathcal{P}(L)\}$. However, we still have to calculate $val(\alpha, \tau)$ for all $\alpha \in S$ and all $\tau \in L$ which we define as follows:

$$val(\alpha,\tau) = \begin{cases} None & \text{if } abs(\tau(i]) \neq \alpha \ \ \forall i \in \{0,...,len(\tau)\}; \\ \Big(avg(\{\hat{\mu}_1(\tau(i_1]),...,\hat{\mu}_1(\tau(i_s])\}), avg(\{\hat{\mu}_2(\tau(i_1]),...,\hat{\mu}_2(\tau(i_s])\})\Big) & \text{otherwise,} \end{cases}$$

where $\{i_1, ..., i_s\} = \{j \in \mathbb{Z} \mid 0 \leq j \leq len(\tau) \text{ and } abs(\tau(i]) = \alpha\}$.

In Table 3 one can obtain the value of $val(\alpha, \tau)$ for all $\alpha \in S$ and $\tau \in L$.

States	Case0	Case1	Case2	Case3	Case4	Case5	Case6	Case7	Case8	Case9
{}	(99.5, 0.775)	(99.5, 0.775)	(99.5, 0.775)	(99.5, 0.775)	(99.5, 0.775)	(99.5, 0.775)	(99.5, 0.775)	(99.5, 0.775)	(99.5, 0.775)	(99.5, 0.775)
{(I, 1)}	(98.5, 0.775)	(98.5, 0.775)	(98.5, 0.775)	(98.5, 0.775)	(98.5, 0.775)	(98.5, 0.775)	(98.5, 0.775)	(98.5, 0.775)	(98.5, 0.775)	(98.5, 0.775)
{(I, 1), (M, 1)}	(60.0, 0.775)	(60.0, 0.775)	(60.0, 0.775)	(60.0, 0.775)	(60.0, 0.775)	(60.0, 0.775)	(60.0, 0.775)	(60.0, 0.775)	(60.0, 0.775)	(60.0, 0.775)
{(I, 1), (M, 2)}	(40.0, 0.82)	(40.0, 0.82)	(40.0, 0.82)	(40.0, 0.82)	None	None	(40.0, 0.82)	None	None	None
{(F, 1), (I, 1), (M, 2)}	(16.25, 0.8)	(16.25, 0.8)	(16.25, 0.8)	None	None	None	(16.25, 0.8)	None	(20.0, 0.6)	None
{(M, 3), (I, 1), (F, 1)}	(7.5, 0.7)	None	None	None	None	None	(7.5, 0.7)	None	(15.0, 0.6)	None
{(M, 3), (I, 1), (F, 1), (S, 1)}	(0.0, 0.7)	None	None	None	None	None	(0.0, 0.7)	None	(0.0, 0.6)	None
{(F, 1), (I, 1), (M, 2), (R, 1)}	None	(5.0, 0.90)	(5.0, 0.90)	(10.0, 0.9)	None	None	None	None	None	None
{(F, 1), (M, 2), (S, 1), (I, 1), (R, 1)}	None	(0.0, 0.90)	(0.0, 0.90)	(0.0, 0.9)	None	None	None	None	None	None
{(I, 1), (M, 2), (R, 1)}	None	None	None	(20.0, 0.9)	None	None	None	(10.0, 0.7)	None	None
{(I, 1), (M, 1), (R, 1)}	None	None	None	None	(38.33, 0.75)	(38.33, 0.75)	None	(38.33, 0.75)	None	None
{(F, 1), (I, 1), (M, 1), (R, 1)}	None	None	None	None	(15.0, 0.775)	(15.0, 0.775)	None	None	None	(15.0, 0.8)
{(M, 1), (F, 1), (S, 1), (I, 1), (R, 1)}	None	None	None	None	(0.0, 0.775)	(0.0, 0.775)	None	None	None	(0.0, 0.8)
{(I, 1), (S, 1), (M, 2), (R, 1)}	None	None	None	None	None	None	None	(0.0, 0.7)	None	None
{(F, 1), (I, 1), (M, 1)}	None	None	None	None	None	None	None	None	(37.5, 0.7)	(37.5, 0.7)

Table 3. Values of $val(\alpha, \tau)$ for all $\alpha \in S$ and $\tau \in L$

In the construction of the PCT, we want to use prepositional logic (or boolean statements) over the values of context features to partition the set of (fully unfolded) traces $\{\tau \in L\}$. The partition of traces into clusters should be made in such a way that the values $\langle val(\alpha_1, \tau), ..., val(\alpha_{15}, \tau)\rangle$ for the traces τ within same cluster (or leaf of PCT) are similar and the same values for traces across distinct clusters (or leaves of PCT) are dissimilar.

In the following, we describe how the variance within each cluster and the weighted variance over many clusters is computed.

First of all, by considering the $val(\alpha, \tau)$-values for all $\tau \in L$, each trace is first represented as a vector of length 15 whose entries are pairs of values (as given in the definition of val). We first convert each vector into one of dimension 30, where the first 15 entries correspond to the first values of the pairs and the second 15 entries to the second ones. Then we normalize each entry into the range $[0, 1]$, so that the remaining time

until submission and the final grade have equal weight when computing the difference between the vectors representing different traces. For the distance function $dist$ we use the euclidian distance. For the $None$ values we set $dist(None, None) = 0$ and $dist(None, x) = 2$ for any $x \in [0, 1]$. This way, the cases where in some position one vector has a $None$ value and the other has a positive defined value is assigned a bigger distance than the cases where two vectors may have very different positive defined values on that same position. Given a set of vectors belonging to some particular cluster, the mean vector v_{mean} (or centroid) of that cluster is a vector of dimension 30 where for each position $i \in \{1, ..., 30\}$ we distinguish between the three cases:
-If all vectors have a $None$ value on position i, then $v_{mean}[i] = None$.
-If all vectors have a positive value on position i, then $v_{mean}[i]$ is set to the average value.
-If only some vectors have a $None$ value in position i, then $v_{mean}[i]$ is set to the average value computed from those vectors having a positive value on that position. This way, there is some kind of penalty for putting together in a cluster vectors where in many positions some have defined values and some not.

The dis(similarity) of the elements in one cluster is calculated as described in Section 4.3 by taking the vector v_{mean} as the centroid and the euclidian distance as $dist$.

Given a list of logical claims $[claim_1 \wedge ... \wedge claim_n]$ regarding the values of the context features, the set of instances is split into two groups, the one containing those who satisfy all claims and the one of those for which at least one claim does not hold. Then the quality of picking these claims as an interior node of the PCT partitioning the set of instances is measured by taking the weigted average variance within cluster as described in Section 4.3.

After experimenting with some claims and measuring their quality, the best clustering is obtained after first splitting the cases based on claim *Field=SE* $\wedge$ *Supervisor=Brown*. This claim holds for cases 4,5,9. The weighted average variance for the partition $\{\{4, 5, 9\}, \{0, 1, 2, 3, 6, 7, 8\}\}$ is approximately 14,19.

We split the set of cases $0, 1, 2, 3, 6, 7, 8$ further by applying the claim *Field=DS* $\wedge$ *PK=high*. This claim holds for cases 1,2,3. The weighted average variance for the partition $\{\{1, 2, 3\}, \{0, 6, 7, 8\}\}$ is approximately 9,52.

Finally, we split the set of cases $0, 6, 7, 8$ further by applying the claim *PK=high*. This claim holds only for case 7. The weighted average variance for the partition $\{\{7\}, \{0, 6, 8\}\}$ is approximately 5,82.

In the end, applying the conditions yielded the partition $\big\{\{4, 5, 9\}, \{1, 2, 3\}, \{7\}, \{0, 6, 8\}\big\}$. The corresponding PCT can be seen in Figure 1.

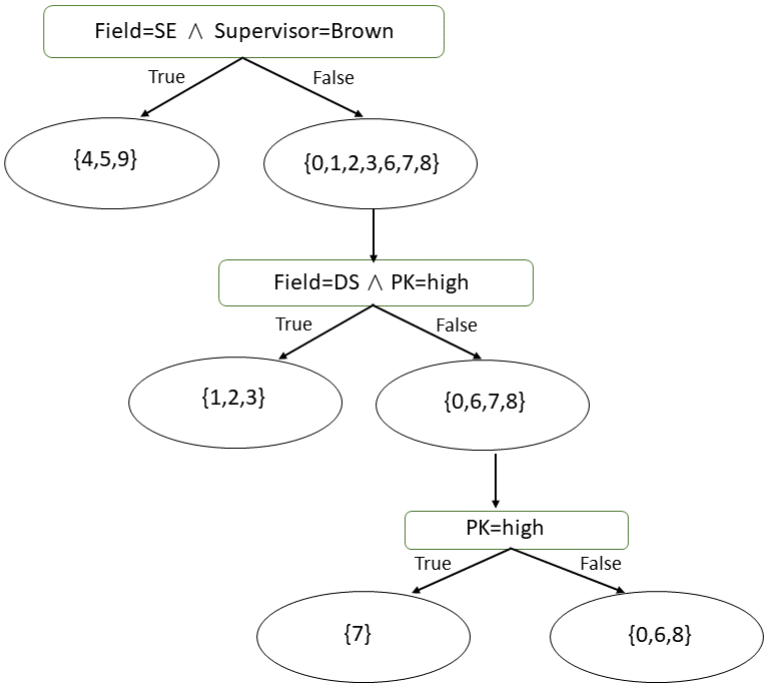

Fig. 1. PCT yielding four clusters. The rectangular nodes contain statements made about the context values.

For any new running case with a particular $context$-value x, its cluster is defined by the following clustering function c:

$$c(x) = \begin{cases} 1 & \text{Field = SE} \wedge \text{Supervisor = Brown} \\ 2 & \text{Field = DS} \wedge \text{PK = high} \\ 3 & \text{PK = high} \\ 4 & \text{otherwise} \end{cases}$$

For the set of states falling into one cluster, we construct the corresponding A-FSM where each state corresponds a multiset of activities (a particular abs_{∞}^{bag} value) and its annotation corresponds to the predicted value $(RT, final\_grade)$ aggregated as average value from the traces falling into the corresponding cluster.

The multiset of values from which the annotation of each state is aggregated can be seen in Tables 4, 5, 6 and 7. The A-FSMs we obtain are the ones in Figures 2, 3, 4 and 5 for clusters 1,2,3 and 4 respectively.

Some valuable knowledge one can extract from the state machines are e.g.: The students who have written their work in the Data Science field and had high previous knowledge (cluster 2) apparently got better grades than the other groups. Also, out of the students who wrote their paper in the Software Engineering field under the supervision of Mr. Brown (cluster 1), the ones who got feedback more early got a slightly better grade and so on.

State	Multiset of predictions	Annotation
{}	[(105, 0.8), (100, 0.75), (95, 0.8)]	(100, 0.78)
{(I, 1)}	[(104, 0.8), (99, 0.75), (94, 0.8)]	(99, 0.78)
{(M, 1), (I, 1)}	[(60, 0.8), (55, 0.75), (55, 0.8)]	(56.67, 0.78)
{(R, 1), (M, 1), (I, 1)}	[(45, 0.8), (30, 0.75)]	(37.5, 0.775)
{(F, 1), (M, 1), (I, 1)}	[(35, 0.8)]	(35, 0.8)
{(R, 1), (F, 1), (M, 1), (I, 1)}	[(20, 0.8), (10, 0.75), (15, 0.8)]	(15, 0.78)
{(M, 1), (R, 1), (S, 1), (I, 1), (F, 1)}	[(0, 0.8), (0, 0.75), (0, 0.8)]	(0, 0.78)

Table 4. Set of states for cluster 1, their multisets of predictions and annotations

State	Multiset of predictions	Annotation
{}	[(105, 0.8), (100, 0.75), (95, 0.8)]	(100, 0.78)
{(I, 1)}	[(104, 0.8), (99, 0.75), (94, 0.8)]	(99, 0.78)
{(M, 1), (I, 1)}	[(60, 0.8), (55, 0.75), (55, 0.8)]	(56.67, 0.78)
{(R, 1), (M, 1), (I, 1)}	[(45, 0.8), (30, 0.75)]	(37.5, 0.78)
{(F, 1), (M, 1), (I, 1)}	[(35, 0.8)]	(35, 0.78)
{(R, 1), (F, 1), (M, 1), (I, 1)}	[(20, 0.8), (10, 0.75), (15, 0.8)]	(15, 0.78)
{(M, 1), (R, 1), (S, 1), (I, 1), (F, 1)}	[(0, 0.8), (0, 0.75), (0, 0.8)]	(0, 0.78)

Table 5. Set of states for cluster 2, their multisets of predictions and annotations

State	Multiset of predictions	Annotation
{}	[(105, 0.8), (100, 0.75), (95, 0.8)]	(100, 0.78)
{(I, 1)}	[(104, 0.8), (99, 0.75), (94, 0.8)]	(99, 0.78)
{(M, 1), (I, 1)}	[(60, 0.8), (55, 0.75), (55, 0.8)]	(56.67, 0.78)
{(R, 1), (M, 1), (I, 1)}	[(45, 0.8), (30, 0.75)]	(37.5, 0.78)
{(F, 1), (M, 1), (I, 1)}	[(35, 0.8)]	(35, 0.78)
{(R, 1), (F, 1), (M, 1), (I, 1)}	[(20, 0.8), (10, 0.75), (15, 0.8)]	(15, 0.78)
{(M, 1), (R, 1), (S, 1), (I, 1), (F, 1)}	[(0, 0.8), (0, 0.75), (0, 0.8)]	(0, 0.78)

Table 6. Set of states for cluster 3, their multisets of predictions and annotations

State	Multiset of predictions	Annotation
{}	[(105, 0.8), (100, 0.75), (95, 0.8)]	(100, 0.78)
{(I, 1)}	[(104, 0.8), (99, 0.75), (94, 0.8)]	(99, 0.78)
{(M, 1), (I, 1)}	[(60, 0.8), (55, 0.75), (55, 0.8)]	(56.67, 0.78)
{(R, 1), (M, 1), (I, 1)}	[(45, 0.8), (30, 0.75)]	(37.5, 0.78)
{(F, 1), (M, 1), (I, 1)}	[(35, 0.8)]	(35, 0.78)
{(R, 1), (F, 1), (M, 1), (I, 1)}	[(20, 0.8), (10, 0.75), (15, 0.8)]	(15, 0.78)
{(M, 1), (R, 1), (S, 1), (I, 1), (F, 1)}	[(0, 0.8), (0, 0.75), (0, 0.8)]	(0, 0.78)

Table 7. Set of states for cluster 4, their multisets of predictions and annotations

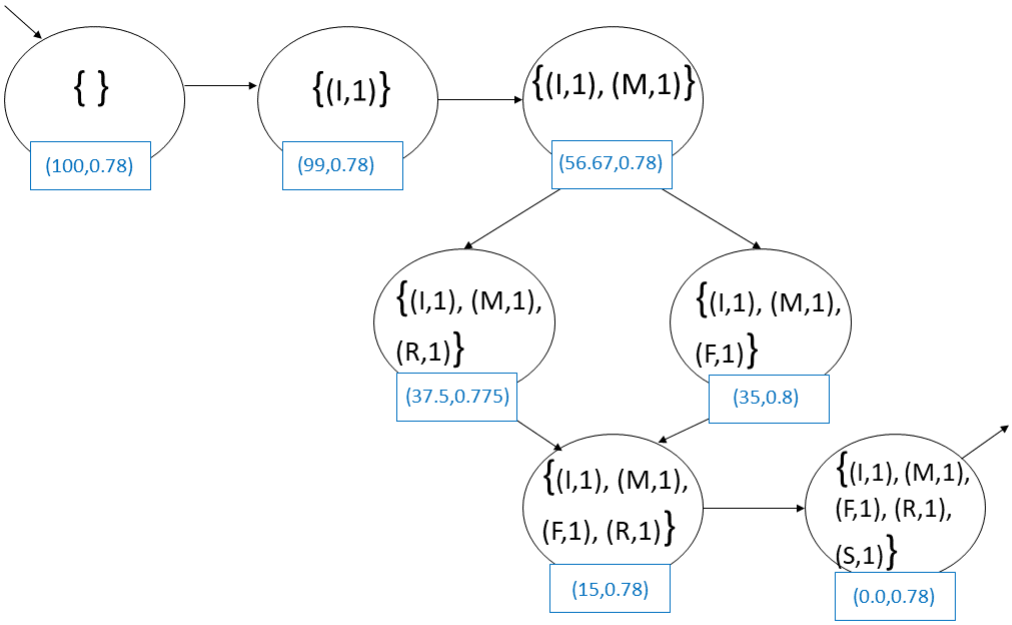

Fig. 2. A-FSM for cluster 1

For each new case, its context will determine the corresponding relevant A-FSM and the annotation of the abstraction of the partial trace related to the case will reveal the prediction.

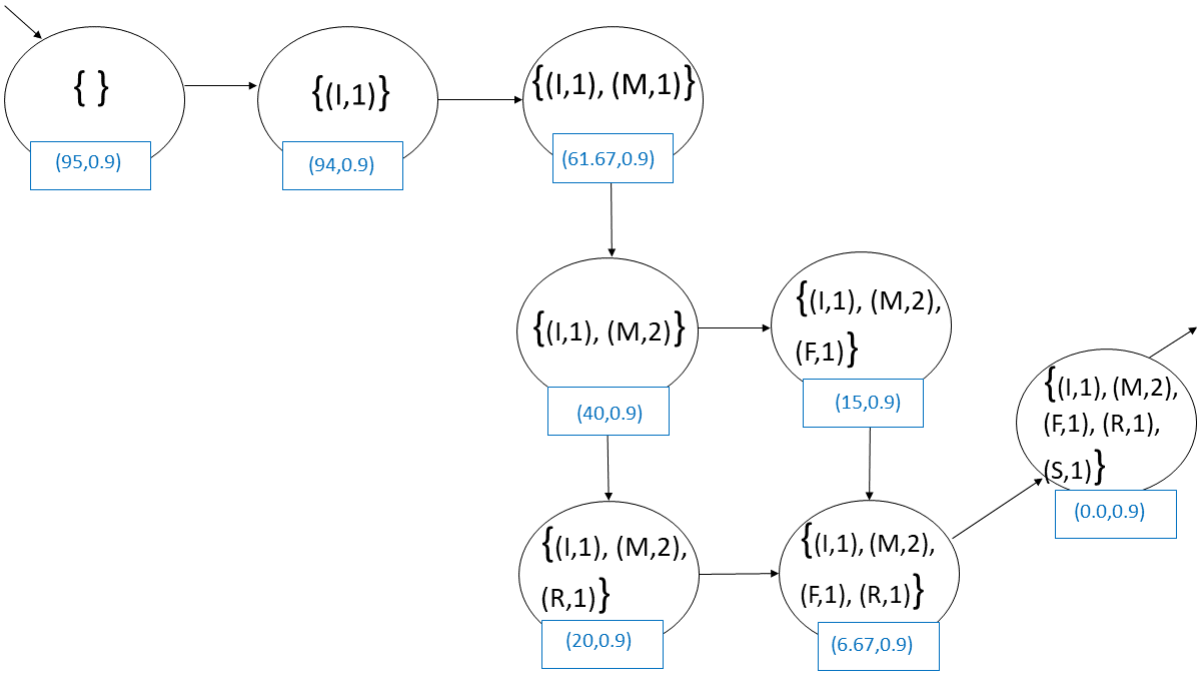

Fig. 3. A-FSM for cluster 2

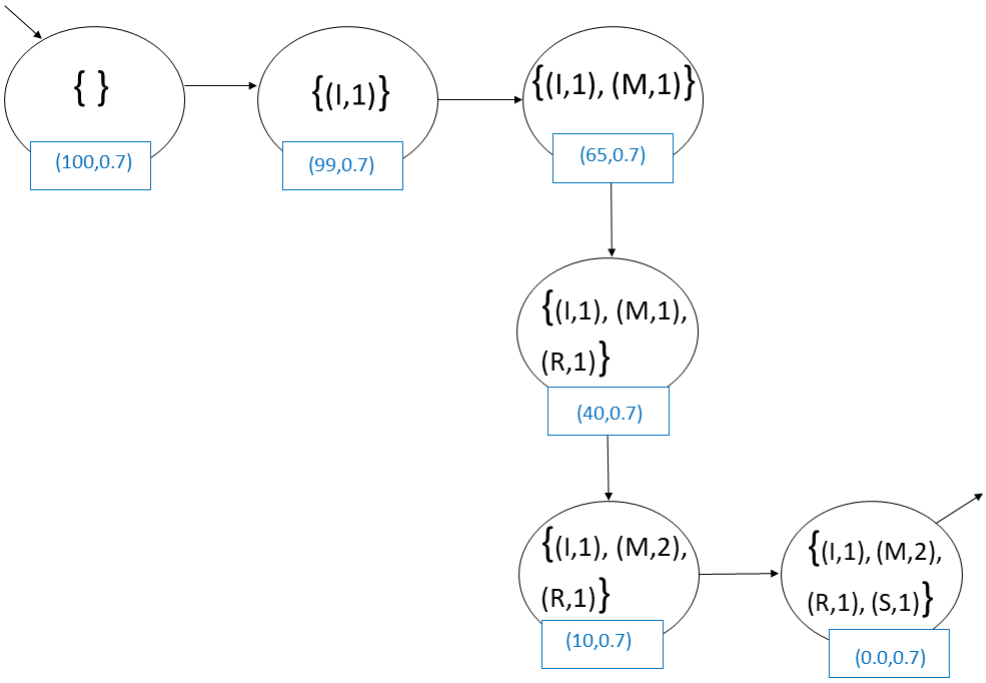

Fig. 4. A-FSM for cluster 3

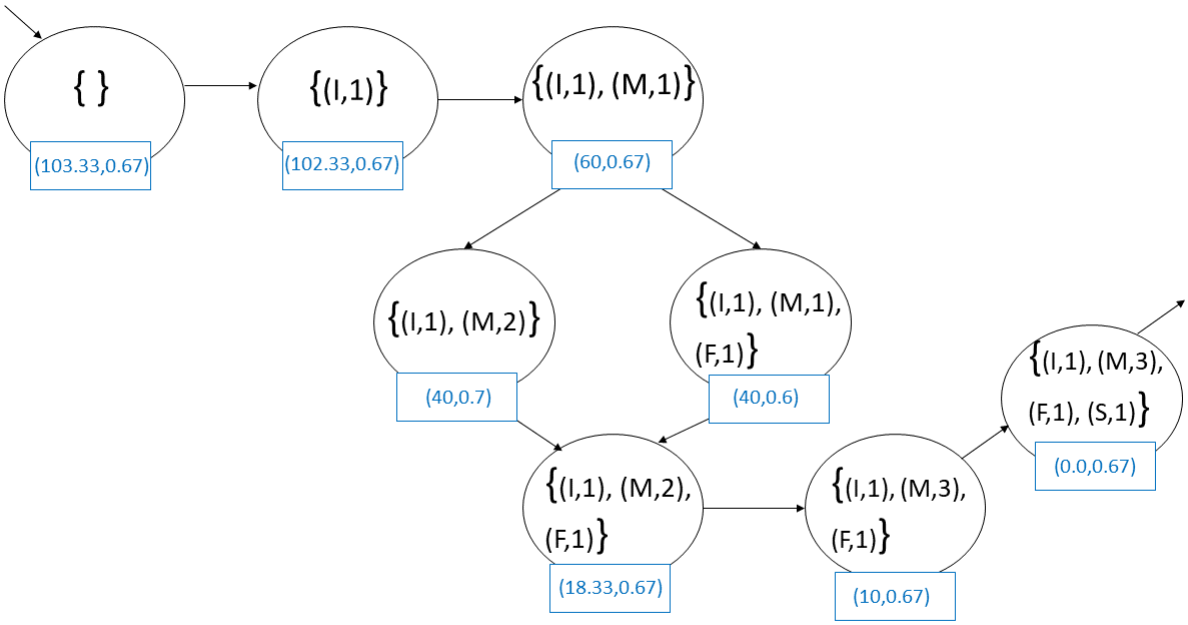

Fig. 5. A-FSM for cluster 4

6 Conclusion

In this work we briefly introduced the approach of constructing Context-Aware Performance Prediction Models which was first described in [1].

The example we gave above is pretty straight-forward and appropriate to understand the mechanisms behing the approach. However, note that the models we obtained are clearly overfitting and do not leave room for predicting the performances of many new process instances. This is mostly due to the fact that the event log is too small and the cases we picked had a similar process flow.

The approach has been implemented as a plug-in in the ProM framework and was validated in a real life case study described in [1]. In the case study, the traces were represented as event labels and the abstraction function only considered the last activity.

Note the approach has a much more general application field in the sense that aspects like representing traces by only considering the labels of the events, picking a simple abstraction function or using the average measure to estimate the annotations are all decisions that can be made depending on what we want to find out about our process, and they are not dictated by the approach.

Moreover, depending on the scneario, one could complement the annotations of the states with comments, explanations or suggestions about possible future actions.

References

1. Francesco Folino, Masssimo Guarascio, Luigi Pontieri.: Discovering Context-Aware Models for Predicting Business Process Performances. Institute for High Performance Computing and Networking (ICAR). R. Meersman et al. (Eds.): OTM 2012, Part I, LNCS 7565, pp.287-304,2012.
2. van der Aalst, W.M.P., Schonenberg, M.H., Song, M.: Time prediction based on process mining. Information Systems 36(2), 450-475 (2011).
3. Blockeel, H., Raedt, L.D., Ramon, J.: Top-down induction of clustering trees. In: Proc. of 15th Intl. Conference on Machine Learning (ICML1998). pp. 55-63 (1998)