Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

# Mining Expressive Performance Models out of Low-level Multidimensional Process Logs

Francesco Folino[1], Massimo Guarascio [1], Luigi Pontieri[1]

RT-ICAR-CS-13-04 Ottobre 2013

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
– Sede di Cosenza, Via P. Bucci 41C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
– Sezione di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.na.icar.cnr.it*
– Sezione di Palermo, Viale delle Scienze, 90128 Palermo, URL: *www.pa.icar.cnr.it*

[1] Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)

# Mining Expressive Performance Models out of Low-level Multidimensional Process Logs

Francesco Folino, Massimo Guarascio, and Luigi Pontieri

National Research Council of Italy (CNR),
via Pietro Bucci 41C, I87036 Rende (CS), Italy
{ffolino,guarascio,pontieri}@icar.cnr.it

**Abstract.** Process Mining techniques have been gaining attention, owing to their potentiality to extract compact process models from massive logs. Traditionally focused on workflows, these techniques tend to rely on a clear specification of process tasks, assumed to be referred explicitly by the logs. This limits however their applicability to many real-life BPM environments (e.g. issue tracking systems) where the traced events do not match any task, but yet keep lots of context data. To make the application of (predictive) process mining to such logs more effective and easier, a novel approach is devised, where the discovery of different execution scenarios is combined with the automatic abstraction of log events. The approach was integrated in a BPA system, also supporting the evaluation of discovered models and OLAP-like analyses. Tested on real-life data, the approach achieved compelling prediction accuracy w.r.t. state-of-the-art methods, and discovered interesting descriptions for both activities and process variants.

**Keywords:** Business Process Analysis, Data Mining, Prediction

## 1 Introduction

The general aim of Process mining techniques [3] is to extract useful information out of historical process logs, possibly providing the analyst with some sort of descriptive (e.g. control-flow oriented) or predictive process model. Such techniques have been given increasing attention in the last decade in the field of Business Process Management, owing to their potentiality to help process analysis and design tasks, and to provide a practical support to process improvement policies. In particular, an emerging research stream [12, 4, 16, 7] concerns the induction of state-aware models for predicting a given performance measure (or KPI), correspondingly to any new process case, which can help integrate advanced run-time support services (e.g., activity recommendation [23] or risk estimation [11]) into the very process enactment or monitoring environment. The basic idea is to exploit log data to induce a state-aware performance model, based on some trace abstraction function (see, e.g., [4, 16]).

Originally focused on workflow systems, the attention of Process Mining researches has been moving towards less structured processes, possibly featuring a wide variety of behaviors, and a high number of tasks. This calls for enhancing classical approaches with the capabilities of capturing diverse execution scenarios (*process variants*) for the process, and of mapping low level tasks to high-level activity concepts [5], so preventing the construction of useless "spaghetti-like" models, providing an overly detailed (and scarcely generalizable) view of process behavior. On the other hand, the need to provide the analyst/designer with high level process views is also witnessed by the multitude of works concerning process abstraction/decomposition (see, e.g., [13, 2, 21, 10]) and the very idea of modelling suitably different process variants and their links to environmental factors [22, 25], and of possibly discovering them automatically via trace clustering methods [24, 15, 20, 16, 7]. In particular, unsupervised log-driven abstraction approaches, [21, 10] try to discover high-level activities directly out of execution logs, by aggregating raw events according to their mutual correlation (estimated on the basis of log evidence). A bunch of approaches was also defined that combine trace clustering and activity abstraction, in order to (semi-)automatically build an expressive representation of the process, showing its main execution variants in terms of high-level activities (sub-processes) [19, 14].

Unfortunately, most of the methods above somewhat rely on a preliminary definition of process tasks, which limits their applicability to many real-life collaborative work environments (such as, e.g., issue tracking systems, or pure transactional systems in general) where the traced execution events (or "audit trail entries") do not refer any predefined process task at all, but rather keep plenty of information, in the form of event/case attributes, which one may well exploit to grasp a high-level description for the performed activities, and, eventually, for the behaviour of the analyzed process.

Moreover, to the best of our knowledge, the combination of model induction and automated activity abstraction has not been investigated adequately in the field of predictive process mining, despite it could help prevent the construction of inaccurate models. In fact, previous proposals [12, 4, 16, 7] mainly reliy on simple expert-driven methods for mapping the given process traces to abstract representations for major execution states of the process, consisting in replacing each log event with the associated task or executor (or them both). However, as in most real logs none of the events' properties fully characterize the semantics of the performed action (and its impact on process performances), an underfitted performance model is likely to be discovered when abstracting the events with just one of their properties. Conversely, if trying to define abstract activities by simply combining the values of multiple event attributes, a cumbersome and yet overfitted model may be generated. In both cases, the discovered models hardly manage to capture the general behavior of the process, and to furnish accurate predictions against unseen process cases, as empirically shown in our tests.

We hence believe that, to effective apply (predictive) process mining to such lowly-structured and yet rich logs, novel automatic log abstraction methods must be devised, accounting for both process performances and process variants.

*Contribution* Conceptually, we face the above issues, we state the problem of inducing a predictive performance model out of a given process log is stated as the search for a new enhanced kind of high-level performance model, featuring three components: *(i)* an event classification criterion (allowing for replacing each low-level log event with a work-item class, corresponding to some unknown process activity, which is frequent, correlated with process performances); *(ii)* a trace classification criterion (discriminating among different process variants, on the basis of case data); and *(iii)* a collection of state-aware predictors, each of them is associated with one of the discovered process variants. Such a model encodes a performance prediction function, where the forecast for the outcome of any new process case $c$ is built in three logical steps: *(i)* first a compact view of $c$'s history is built by replacing each event occurred in its partial enactment with an event class (based on the first function); *(ii)* the abstract trace so produced is then assigned to process variant (case class) via the second function, and *(iii)* the corresponding predictor of is eventually used to make the forecast.

Technically, we devise an algorithm for inducing, out of a given log, such a multi-facet process model, where the two basic classification functions (relative to cases and events, respectively) are learnt through an iterative co-clustering scheme, leveraging and extending the consolidate framework of (logics-based) *predictive clustering* [8], before equipping each trace cluster with a local predictor (like those used in [7]). By inducing a definition of both abstract activities of process variants, both expressed in terms of logical decision rules, the approach suits well the difficult process mining settings mentioned before (complex processes, with different context-dependent variants, and low-level logs), while freeing the analyst from the need to explicitly define a mapping between log records and high level process tasks — indeed, our (log-driven and performance-aware) unsupervised classification of log events allows for mapping them to relevant activity patterns, expressed at the right abstraction level as far as concerns the characterization (and prediction) of performance behaviors. Besides enjoying satisfactory prediction accuracy (often better than current methods with usual event abstraction criteria) in real application scenarios, the descriptive nature of the discovered events' and traces' classification rules (expressed in terms of their associated data properties) helps the analyst comprehend process behaviors, and the way its performances depends on both context factors and activity patterns.

An innovative Business Process Analysis architecture is also presented, which supports the analyst to discover of such a multi-perspective performance model (by way of the proposed learning approach), as well as to evaluate and reuse the discovered knowledge, and to embed it into advanced monitoring mechanisms.

*Organization* The rest of the paper is structured as follows. After introducing some preliminary concepts and notation (in Section 2), we formally state the problem faced in the paper, and illustrate a learning algorithm for solving it (in Section 3). The Section 4 presents the prototype system implemented. We then discuss, in Section 5, an empirical analysis conducted on two real-life case studies, allowing for assessing the validity of the proposed approach and the practical usefulness of the models discovered. A few concluding remarks and future work directions are drawn in Section 6.

## 2 Preliminaries

*Process Logs* As usual, we assume that a *trace* is recorded for each process instance (a.k.a "case"), storing the sequence of *events* happened during its enactment. Let $E$ and $\mathcal{T}$ be the universes of all possible events and traces, respectively, for the process under analysis. An event $e \in E$ can be regarded as a tuple $\langle cID, t, x_1, \ldots, x_n \rangle$, where $cID$ is the identifier of a process instance, $t$ is a timestamp, while $x_1, \ldots, x_m$ is a list of data values corresponding to some given event attributes $X_1^E, \ldots, X_m^E$, respectively. For the sake of conciseness, let $prop(E)$ denote the space of all these latter attributes (i.e., $prop(E) = X_1^E \times \ldots \times X_m^E$), and $prop(e) = \langle X_1^E, \ldots, X_m^E \rangle$ be the (sub-)tuple of all data associated with any event $e \in E$; moreover, let $case(e)$ and $time(e)$ denote the case ID and timestamp associated with $e$, and let $X_i^E(e)$ be the value taken by attribute $X_i^E$ on $e$, for $i = 1, \ldots, m$. For example, in structured process management settings (like those handled by way of a WfMS) each event refers to both a *task* and an executor ("resource") — which could be denoted by $task(e)$ and $executor(e)$, respectively. This does not happen, however, in more flexible collaboration environments (e.g., those used in the areas of ticket/issue handling, bug tracking, project management, and document versioning), where no precise conceptualization of process tasks is available, or these just correspond to very generic kinds of operations (e.g., updating a document/field and/or exchanging a message). And yet, many data attributes can be stored for an execution event (such as, e.g., data parameters), which can help infer the semantics of the corresponding activity. In fact we just look at log events as a precise snapshot of the executed work items, without assuming that these are necessarily modeled in terms of well specified process tasks, so trying to extend the scope of application of Process Mining techniques to a broader range of BPM scenarios than the workflow-based one they were originally devised for.

For each trace $\tau \in \mathcal{T}$, let $len(\tau)$ be the number of events stored in $\tau$, $\tau[i]$ be the $i$-th event of $\tau$, for $i = 1 \mathinner{..} len(\tau)$, and $\tau(i] \in \mathcal{T}$ be the *prefix* trace consisting only of the first $i$ events in $\tau$, for any $i = 0, \ldots, len(\tau)$. Besides the mere sequence of registered events, most tracing systems keeps the values of a number of data attributes, say $X_1^{\mathcal{T}}, \ldots, X_m^{\mathcal{T}}$, for each process instance. As for the case of events, let us denote $prop(T) = X_1^{\mathcal{T}} \times \ldots \times X_m^{\mathcal{T}}$, and let $con(\tau)$ be the tuple storing the values associated with any trace $\tau \in \mathcal{T}$. Notice that the properties of both events and traces can be extended by adding a series of "environment" variables (such as, e.g., workload indicators, temporal dimensions), capturing the state of the surrounding enactment system, in addition to their associated "intrinsic" properties (as also proposed in [16]).

A *log* $L$ (over $\mathcal{T}$) is a finite subset of $\mathcal{T}$, while the *prefix set* of $L$, denoted by $\mathcal{P}(L)$, is the set of all prefix traces that can be extracted from $L$, and $events(L)$ is the set of all events stored in (some trace of) $L$.

*Performances measures and basic prediction approach.* Let us denote by $\lambda : \mathcal{T} \rightarrow \mathbb{R}$ the (unknown) performance measure that is the target of our analysis, virtually assigning a performance value to any (possibly partial) trace – for the sake of concreteness, and w.l.o.g., we assume that $\lambda$ ranges over real numbers. Two notable examples of such a function are the remaining processing time and steps, denoted by $\lambda^{RT}$ and $\lambda^{RS}$, which return the time and steps, respectively, that are still needed to complete the process instance associated with the trace.

A (predictive) *Process Performance Model (PPM)* is a model that can estimate the unknown performance value of a process enactment, based on the contents of the corresponding trace. Such a model can be viewed as a function $\lambda' : \mathcal{T} \rightarrow \mathbb{R}$ approximating $\lambda$ 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 $\lambda(\tau)$ of the target measure is known for each (sub-)trace $\tau \in \mathcal{P}(L)$ — in fact, for each $\tau$ and $\tau(i]$, it is $\lambda^{RT}(\tau(i]) = time(\tau[len(\tau)]) - time(\tau[i])$ and $\lambda^{RS}(\tau(i]) = len(\tau) - i$.

Current approaches to this problem [4, 12, 16] rely all on the idea of applying some suitable abstraction function to process traces, allowing to focus on those facets of the registered events that influence the most process performances.

An *event abstraction function* $\mathcal{E}$ is a function mapping each event $e \in E$ to an abstract representation $\mathcal{E}(e)$. Based on such a function, we can define the abstracted view of an entire trace, in order to provide a more concise encoding for its actual sequence of events, as well as for its context data. A common approach to defining such a view consists in simply regarding the trace as a multi-set (a.k.a. bag) of abstracted events.

**Definition 1 (Trace Abstraction).** Let $\mathcal{T}$ be a trace universe and $\mathcal{E}$ be an event abstraction function, mapping each event in $E$ to an abstract event in $\{\hat{e}_1, \ldots, \hat{e}_k\}$. Then, for each trace $\tau \in \mathcal{T}$, the trace abstraction function $abs^{\mathcal{E}} : E \rightarrow \mathbb{N}^n$ is defined as follows: $abs^{\mathcal{E}}(\tau) = \langle count(\hat{e}_1, \tau), \ldots, count(\hat{e}_k, \tau)\rangle$, where $count(\hat{e_i}, \tau) = |\{\ i \in \{1, \ldots, len(\tau)\} \mid \mathcal{E}(\tau[i]) = \hat{e}_i\ \}|$. $\square$

In this way, each trace $\tau$ is summarized into a tuple, each component of which corresponds to a single abstract activity, and stores the number of times this latter occurs in $\tau$. Notice that the above trace abstraction criterion is similar to those used in the literature [4, 16] — to be precise, these latter works also considered the possibility to turn a trace into a list (or set) of abstracted events, while possibly focusing on the most recent ones, based on a horizon threshold.

*Example 1.* Let us consider a real-life case study pertaining a transshipment process, also used in the tests of Section 5. The main attribute of each trace event concerns the kind of move operation (`moveType`) performed, which may include the following ones: transferring it from one slot to another ($MOV$), charging it onto a ship ($OUT$), and swapping it with another container ($SHF$). Other relevant event attributes are: the shift when the operation was done, the vehicle used (e.g., straddle carrier, multi-trailer, crane), and source/destination position — expressed at different granularities, in terms of yard slots, blocks, and areas. To apply traditional approaches to such data, one can define an event abstraction function $\mathcal{E}^{MT}$ that simply replaces each event with the associated `MoveType`, hence regarded as an atomic task of the logistics process. For example, let us assume that $\tau$ is a log trace encoding the sequence $\langle e_1, e_2, e_3 \rangle$ of three events, with $\texttt{MoveType}(e_1) = \texttt{MoveType}(e_2) = MOV$ and $\texttt{MoveType}(e_3) = OUT$. Using the function in Def. 1, the prefixes $\tau(1]$, $\tau(2]$ and $\tau(3] \equiv \tau$ (i.e. the enactments ending at the first, second and third event, respectively) are turned into three tuples, encoding the multi-sets $[MOV]$, $[MOV^2]$, and $[MOV^2, OUT]$, respectively. $\lhd$

Based on such trace abstractions, a PPM can be derived as proposed in the pioneering work [4], in terms of an annotated finite state machine (named "AFSM"), where each node corresponds to one abstract trace representation (produced by $abs^{\mathcal{E}}$) and stores an estimate for the target measure (usually computed as the average over all trace prefixes reaching that state), while each transition is labelled with an event abstraction (produced by $\mathcal{E}$).

*Predictive clustering and performance prediction* In general, the core assumption of Predictive Clustering [8] is that, based on a suitable clustering model, predictions for new instances can be based on the cluster where they are estimated to belong. In such a setting, it is assumed that two kinds of features are available for any element $z$ in the given instance space $Z = D \times Y$: *descriptive* features and *target* features (i.e., the ones to be predicted). Then, the task amounts to induce a (predictive clustering) function $m : D \rightarrow Y$ of the form $m(x) = p(c(d), d)$, such that $c : D \rightarrow \mathbb{N}$ is a partitioning function and $p : \mathbb{N} \times D \rightarrow Y$ is a (cluster-based) prediction function. Clearly, when dealing with multiple target features, $q$ is to encode a multi-regression model. Several PCM learning methods have been proposed in the literature, which can work with general relational data [8], or with propositional data only (e.g., system CLUS [1]).

The application of predictive clustering techniques to log data, as a solution for predicting process performance, was originally proposed in [16], based on the belief that process performances may depend on context data, which can be then used as descriptive variables for partitioning the log, while using a number of associated performance measurements as target. To this end, a propositional encoding of log traces and of their associated performance values is used to induce such a partitioning function, and to eventually derive a PPM from each of the discovered clusters. A similar approach was used in [7], using classic regression methods for propositional data as an alternative to AFMS models. Such a solution was proven quite effective and scalable, and easier to use, in that it allows to find the right abstraction level over the sequence of events in a data-driven way (without requiring the analyst to suitably tune the horizon threshold).

## 3 Problem Statement and Approach

After introducing a more precise formulation of the problem that we want to face, we next illustrate our solution approach.

### 3.1 Problem Statement

Clearly, the effectiveness of current performance mining approaches strongly depends on the capability of the event abstraction function $\mathcal{E}$ to focus on facets of log events that are really correlated with the behavior of the process and, in particular, with its typical performance patterns. Unfortunately, the common solution of abstracting each event into the associated task or executor (or them both), does not fit many real logs, consisting of fine grain records, which only correspond to low level, generic, operations (and conveying little information on the semantics of the activities performed), or even lack any conceptualization of process tasks. In such a situation, none of the events' properties may be suitable for abstraction purposes, in that it is not sufficient to fully capture relevant activity execution patterns (which are both frequent and correlated to performance outcomes). For instance, with regard to Example 1, when only looking at the kind of move performed in a processing step, several other properties of it (such as the transportation means, or the involved yard positions) are disregarded, which might have helped discover a more refined model of process performances. However, defining abstract activities by combining the values of multiple event properties, as proposed in [4], may well yield a cumbersome representation (with a combinatorial number of trace features and, hence, of process states), which risks being ineffective in many application scenarios, as shown by our tests.

Beside exhibiting a far richer representation of events, w.r.t. traditional workflow logs, many real application contexts are also characterized by the presence of lots of context information for each process case. For example, in the above transshipment system, several physical properties (e.g., size and weight) are kept for each container, as well as its previous and next call, and the navigation lines delivering/loading it. By also exploiting such context factors, a more precise and articulated performance prediction model. In particular, we still rely on the idea (introduced in [16]) of using these trace attributes as descriptive features for discriminating among different context-related execution scenarios.

In order to fully exploit the variety of data stored in a process log, we attempt to build a high-level and modular performance model for the process, based on two interrelated classification models: one allowing to grasp the right level of abstraction over log events, and the other encoding the business rules that underly each of its variants. A precise definition of such a model is given below.

**Definition 2 (`CCPM`).** Let $L$ be a log, over some given event universe $E$ and trace universe $\mathcal{T}$. Then, a *Co-Clustering Performance Model* (`CCPM`) for $L$ is a triple of the form $M = \langle \mathcal{C}_E, \mathcal{C}_T, \Lambda \rangle$, where: *(i)* $\mathcal{C}_E : E \to \mathbb{N}$ is a partitioning function over $E$; *(ii)* $\mathcal{C}_T : \mathcal{T} \to \mathbb{N}$ is a partitioning function over $T$; and *(iii)* $\Lambda = \langle \lambda_i, \ldots, \lambda_q \rangle$ is a list of performance prediction models, all using $\mathcal{C}_E$ as event abstraction function, where $q$ is the number of clusters produced by $\mathcal{C}_T$, and $\lambda_i$ is the model of the $i$-th cluster, for $i = 1 \ldots q$. The overall prediction function encoded by $M$ (also denoted by $M$, for shortness) is: $M(\tau) = \lambda_j(\tau)$, where $j{=}\mathcal{C}_{\mathcal{T}}(\tau)$. □

Conceptually, a forecast for any new process instance $\tau$ can be made in three steps with the help of such a model: *(i)* first an abstract version of $\tau$ is obtained in the form of a vector (as specified in Def. 1) summarizing both its context data and structure, where each event is abstracted into an event class via $\mathcal{C}_E$; *(ii)* $\tau$

is then assigned to a trace cluster (representing a particular execution scenario for the process) via function $\mathcal{C}_{\mathcal{T}}$; *(iii)* the predictor of this cluster is finally used to make a forecast for $\tau$, by providing it with $abs^{\mathcal{C}_E}(\tau)$.

The functions $\mathcal{C}_E$ and $\mathcal{C}_T$, encoding two different classification models, are hence exploited to abstract raw log event into high-level classes, and to discriminate among different process variants based on context data, respectively.

Notably, in our setting, the very definition of an event abstraction criterion becomes a key part of an induction problem, amounting to learn the above specified kind of prediction model (capturing the hidden performance measure), based on a given log of historical traces. This frees the analyst from the burden of finding the right abstraction level over events, which is instead a preliminary delicate task for current approaches. On the other hand, the automated recognition of multiple process variants helps obtain more precise forecasts than a single PPM model, especially when working with complex and/or flexible processes.

### 3.2 A solution approach to $CCPM$ discovery: Algorithm CCD

In principle, one might search for an optimal CCPM for the given log $L$, as the one minimizing some suitable loss measure, contrasting the actual performance of each trace to the corresponding prediction (possibly using an independent test sample). By contrast, to avoid prohibitive computation times (due to the large number of instances that may exist for functions $\mathcal{C}_E$ and $\mathcal{C}_T$ in many contexts), we turn the problem into two simpler ones: *(i)* find a locally optimal pair of classification functions $\mathcal{C}_E$ and $\mathcal{C}_{\mathcal{T}}$, and *(ii)* derive a collection of full PPM predictors, one for each trace cluster, according to them.

Our solution approach is illustrated in Figure 1, in the form of an algorithm, which is named CCD (standing for "Co-Clustering based Discovery"). Since the quality of the trace clustering model $\mathcal{C}_T$ strongly depends on the chosen abstraction function $\mathcal{C}_E$, and vice versa, we regard the first subproblem as a multi-domain clustering (i.e. co-clustering) problem, were an optimal partition must be find for both the traces and the events. This problem is approached via an iterative alternate-optimization scheme (somewhat resembling that of numerical bi-clustering and matrix approximation methods [6]), where, at each iteration $k$, the updated versions of the two partitioning functions are computed, denoted by $\mathcal{C}_E^{(k)}$ and $\mathcal{C}_T^{(k)}$, till a satisfactory precision improvement is achieved, with respect to the previous iteration.

For the sake of efficiency, prediction errors are estimated through an approximated loss measure, denoted by $Err^{(k)}$ for any iteration $k$, which only accounts for the distribution of performance values within each "co-cluster" (i.e., each pair of trace cluster and event cluster), without requiring the computation of a complete $CCPM$. By way of a predictive clustering method, each model $\mathcal{C}_E^{(k)}$ is induced from a propositional "event-oriented view (named *e-view*) $EV$ of the input log, which is meant to capture the relationship between event data and different performance values, computed in correspondence of all current trace clusters, i.e. the range of function $\mathcal{C}_T^{(k-1)}$. The discovered event clustering $\mathcal{C}_E^{(k)}$

```
Input: A log $L$ (over some trace universe $\mathcal{T}$) with an associated target measure $\lambda$,
    max. iterations' number $maxIter \in \mathbb{N}$, min. (relative) loss reduction $\gamma \in (0,1]$,
    max. number $maxCl_E, maxCl_T \in \mathbb{N} \cup \{\infty\}$ of (events', resp. traces') clusters,
    min. cluster coverages $\sigma \in (0,1]$ (for both events and traces).
Output: A CC-PPM model for $L$ (fully encoding $\lambda$ all over $\mathcal{T}$).
Method: Perform the following steps:
 1 set $\hat{T}^{(0)} := \{L\}$;  $\hat{E}^{(0)} := \{events(L)\}$;  $Err^{(0)} := \infty$;  $k := 0$;
 2 do
 3    k := k+1;
 4    $EV := \mathcal{V}_E(L, \mathcal{C}_T^{(k-1)})$; // build an e-view for L w.r.t. $\mathcal{C}_T^{(k-1)}$ (cf. Def. 3)
 5    $\mathcal{C}_E^{(k)} :=$ minePCM($EV, \sigma, maxCl_E$) ; // induce a novel event clustering model
 6    $TV := \mathcal{V}_T(L, \mathcal{C}_E^{(k)})$; // build a t-view for L w.r.t. $\mathcal{C}_E^{(k)}$ (cf. Def. 4)
 7    $\mathcal{C}_T^{(k)} :=$ minePCM($TV, \sigma, maxCl_T$) ; // induce a novel trace clustering model
 8    let $Err^{(k)} = Loss(\mathcal{C}_E^{(k)}, \mathcal{C}_T^{(k)}, L)$; // estimate current prediction error
 9    $improved := Err^{(k-1)} - Err^{(k)} \leq \gamma^{(k)} \times Err^{(k-1)}$;
10 while $k \leq maxIter$ and $improved$;
11 if $improved$ then  $\mathcal{C}_T := \mathcal{C}_T^{(k)}$;  $\mathcal{C}_E := \mathcal{C}_E^{(k)}$;
12                 else  $\mathcal{C}_T := \mathcal{C}_T^{(k-1)}$;  $\mathcal{C}_E := \mathcal{C}_E^{(k-1)}$;
13 let $TC = \langle \hat{t}_1, \ldots, \hat{t}_q \rangle$ be the list of trace clusters produced by $\mathcal{C}_T$ on $\mathcal{P}(L)$;
14 for each $\hat{t}_i$ in $TC$ do
15    $\lambda_i :=$ minePPM($\hat{t}_i, \mathcal{C}_E$);
16 end
17 return $\langle \mathcal{C}_E, \mathcal{C}_T, \langle \lambda_1, \ldots, \lambda_q \rangle \rangle$
```

**Fig. 1. Algorithm** `CCD`**.**

are then used, as a novel event abstraction function, to produce a summarized propositional "trace-oriented view (named *t-view*) $TV$ of the log, which is again given as input to a predictive clustering algorithm, to discover an updated trace partitioning model $\mathcal{C}_T^{(k)}$. In this way, any novel trace clustering takes advantage of the most recent definition of the event classes (abstract activities), and vice versa, according to a reinforcement learning perspective. Both event clusters and trace clusters are discovered via function `minePCM` ("mine Predictive Cluster Model"), which implements a predictive clustering method, allowing to induce a partitioning function for one or multiple target attributes, out of a given propositional dataset, featuring a number of additional descriptive attributes (the ones that will be used in partitioning function) — in fact, this is right the structure of both the e-view $EV$ and the t-view $TV$, as explained in details later on. Two auxiliary parameters are taken by this function: a minimal coverage percentage for each discovered cluster, and an upper bound for the number of clusters, respectively.

Once an (locally) optimal pair of event and trace clustering functions have been found, each local cluster predictor ($\lambda_i$, for $i = 1..n$) is eventually computed through function `minePPM` ("mine Process Performance Model"), provided with the set of traces assigned to the clusters, and with a suitable event abstraction function — which, in our case, corresponds to the event partitioning function $C_E$, computed at the end of the co-clustering loop, which right allows for replacing each event (based on its associated attributes) with an event cluster (regarded as an abstract activity pattern). To this end, each cluster $\hat{t}_i$ is preliminary converted into table, by generating its t-view (with $\mathcal{C}_E$ used for event abstraction).

Before providing the reader with technical details on *e-views*, *t-views* and the loss measure, we notice that (besides process traces and associated target performances) the algorithm can be provided with a series of parameters, giving the analyst some control on the structure of the discovered model, as well as on the computation time. We pinpoint that the approach does not require big efforts for the tuning of these parameters, and performs well over a wide range of settings, as discussed in detail in Section 5.6. In fact, $maxCl_E$ and $maxCl_T$ mainly help discover compact and readable models, while $maxIter$ and $\gamma$ can allow for speeding up the computation. Notably, in practical cases, the computation tends to converge naturally in a few steps, without using restrictive stopping criteria, so that one might even use the most liberal configuration for both $\gamma$ $(= 0)$ and $maxIter$ $(= \infty)$. However, it should be safer to fix, at least, a reasonable bound (e.g., 20) for the iterations' number, and some low positive value for $\gamma$, for the sake of robustness to numerical approximation.

Anyway, no matter of input parameters, the algorithm always converges to a local minimum for the loss function, seeing as this latter must decrease its value at each iteration w.r.t. the previous one — observe indeed that it is required that $\gamma > 0$— and the number of clustering functions for both events and traces is finite. Moreover, since the novel clusterings are chosen via the same greedy deterministic procedure, it is likely that the stop condition is met in a few iterations — as confirmed by our empirical analysis on real data.

*Technical details: E-Views, T-Views, and Loss* Both kinds of log views, as well as the error measure used in the optimization loop, are formally defined next.

**Definition 3 (E-View).** Let $L$ be a log, with associated event and trace universes $E$ and $\mathcal{T}$. Let $\mathcal{C}_T$ be a given (trace) partitioning function defined over $\mathcal{P}(L)$, and let $\{\hat{t}_1, \ldots, \hat{t}_q\}$ be its associated clusters — i.e., the range of $\mathcal{C}_T$. Then, an *e-view* ("event-centric view") for $L$ w.r.t. $\mathcal{C}_T$, denoted by $\mathcal{V}_E(L, \mathcal{C}_T)$, is a relation consisting of a tuple $z_e = prop(e) \| \langle val(e, \hat{t}_1), \ldots, val(e, \hat{t}_q) \rangle$ for each $e \in events(L)$, with $\|$ standing for tuple concatenation, such that, for $i = 1, \ldots, q$, it is:

$$val(e, \hat{t}_i) = \begin{cases} \texttt{NULL}, \text{if } \nexists\ \tau \in \hat{t}_i^{(k)} \text{ s.t. } prop(\tau[len(\tau)]) = prop(e); \\ avg(\{\lambda(\tau) \mid \tau \in \hat{t}_i^{(k)} \text{ and } prop(\tau[len(\tau)]) = e\ \}), \text{ otherwise.} \end{cases}$$

With regard to predictive clustering, for any such tuple $z_e$, all the fields in $prop(e)$ are to be considered as descriptive variables, and $\langle val(e, \hat{t}_1), \ldots, val(e, \hat{t}_q) \rangle$ as the associated (multidimensional) target. □

Such a view of the log is intended to serve as a training set in the (unsupervised) learning of an event classification function, with the help of some predictive clustering procedure. Clearly, the discovered event classification will depend on the given current trace clusters, and, in particular, on how the performance values associated with $prop(e)$ are distributed across them.

**Definition 4 (Trace View).** Let $L$ be a log, with associated event and trace universes $E$ and $\mathcal{T}$. Let $\mathcal{C}_E$ a given (event) partitioning function defined over $events(L)$, and ranging over the clusters $\{\hat{e}_1^{(k)}, \ldots, \hat{e}_p^{(k)}\}$. Then, a *t-view* ("trace-centric view") for $L$ w.r.t. $\mathcal{C}_E$, denoted by $\mathcal{V}_T(L, \mathcal{C}_E)$, is a relation containing, for each (partial) trace $\tau \in \mathcal{P}(L)$, a tuple $z_\tau = prop(\tau) \parallel abs^{\mathcal{C}_E}(\tau) \parallel \langle \lambda(\tau) \rangle$, where $\parallel$ still denotes tuple concatenation. For any such tuple $z_\tau$, $prop(\tau)$ and $abs^{\mathcal{C}_E}(\tau)$ are considered as descriptive features, and $\lambda(\tau)$ as the (scalar) target. $\square$

Similarly to the previous case, a trace sketch is exploited to learn a trace classification model by way of some suitable predictive clustering procedure, where the context data and the structural abstraction of each trace are used as descriptive features, allowing to split the (partial) traces into homogenous groups with respect to their associated performance values. In this way, it is possible to exploit all information stored in any partial trace, be it the collection of abstract activities (captured by event classes $\mathcal{C}_E$) appearing in it, or the array of its associated (updated) context data — differently from [16], where each training instance represents a fully unfolded trace, using only the initial values (i.e., registered at the start of the process) of its associated data properties as descriptive features.

## 4 Implementation: an Advanced BPA Prototype System

The learning approach described so far has been integrated into a prototype system, aimed at providing advanced services for the analysis and monitoring of process performances. The system features a three-layer conceptual architecture, as shown in Figure 2. The lowest layer, the *Data/Model Repository Layer*, is responsible for storing both historical process logs and different kinds of propositional sketches derived from them prior to each induction task. The *Knowledge Discovery* layer encapsulates some core functionalities for building and handling different kinds of models, representing different facets of process behavior: *(i) Event Clustering Models*, providing an effective, data-driven, way to abstract raw events into high-level activities, *(ii) Trace Clustering Models*, capturing diverse execution scenarios for the analyzed process, and their correlation with major context factors, and *(iii)* a series of performance prediction models ($PPM$s) (one for each discovered scenario).

All these core models are discovered, in a interactive and iterative manner, based on the computation scheme of algorithm `CCD` — possibly instantiated multiple times, with different (random) initializations of both events' and traces' clusters, in order to possibly explore several local optima, and eventually keep the best of them. More specifically, the *Log Processing* module is in charge of generating the e-views and t-views for a given log (cf. Definitions 3 and 4). Optionally, the *Log Processing* module can enrich each event/trace instance with extrinsic (environmental) context features computed on the fly, such as workload

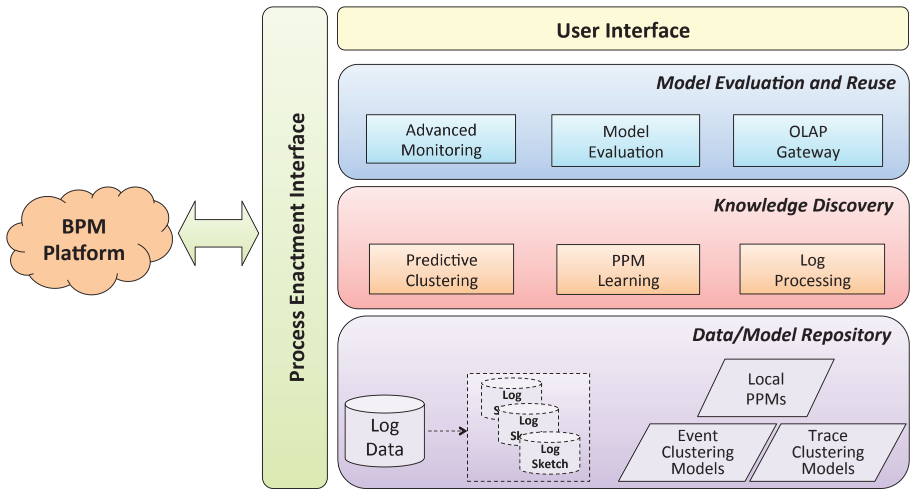


**Fig. 2.** Conceptual architecture of the developed prototype system.

indicators and aggregated time dimensions, by taking advantage of auxiliary data structures for efficiently looking up the neighborhood of any log' trace/event.

The *Predictive Clustering* and *PPM Learning* modules are mainly devoted to compute the `minePCM` and `minePPM` functions, respectively, called in the algorithm of Figure 1. Currently, the implementation of the former function leverages an algorithm for the induction of a PCT (Predictive Clustering Tree) [8, 9], a logics-based predictive clustering model, where the cluster assignment function is encoded in terms of decision rules (over descriptive attributes). Such a model is built via a top-down partitioning scheme, where the log is split recursively, by selecting every time a descriptive attribute that locally minimizes the variances of the newly generated clusters. An F-test based stopping criterion is used to curb the growth of the tree, possibly combined with an user-given upper bound on the total number clusters — used as size constraint in the application (to the fully grown tree) of the pruning procedure in [18].

Log traces, with the events abstracted into their respective classes, are then delivered to the *PPM Learning* module, which derives one *PPM* for each trace cluster, through one of the following prediction techniques: the tree-based regression algorithm *RepTree* [26], and the instance-based *IBK* [26]. Notice that we are working on integrating other numeric prediction algorithms, to enlarge the range of alternative PPM learners offered to the analyst.

For the sake of further analysis, all the models discovered out of a process log (i.e., traces' and events' clustering models, and the PPM models of each trace cluster) are made available to the *Model Evaluation and Reuse Layer*. All these models can be explored with the help of the *Model Evaluation* module, which, in particular, provides the user with an easily-readable report, including the classical error metrics (e.g., *rmse*, *mae*, and *mape*) introduced in Section 5.

The *OLAP Gateway* module is meant to reorganize historical log data into different aggregated forms, in order to possibly apply OLAP-like methods. More specifically, the module allows the analyst to dynamically compute aggregated statistics (Figure 4), as well as to present data-driven aggregation hierarchies for a process/event attribute, derived automatically from trace/event clusters (Figure 3). The latter kind of result is obtained by applying a hierarchical clustering algorithms to a contingency matrix representing in how many clusters two values of the analyzed attribute occur together. Clearly, this capability is particularly interesting for attributes with high cardinality, as it allows the analyst to get a summarized representation of the attribute, and can support the design of a datacube for analysing process performances, Even though such consolidated data could be efficiently processed with the help of an OLAP engine, the actual implementation of the system only works with virtual aggregated views (computed on-the-fly), without precomputing them.

Thanks to its predictive nature, each `CCPM` model can be used to configure a forecasting service for the process it was discovered for, to estimate (at run time and step-by-step) the performance outcome of any new instance of that process. Such a service (taking as input the partial trace of an ongoing process enactment, and returning a forecast for the associated measure, such as the remaining processing time/steps), can be accessed through the *Process Enactment Interface*, by any process enactment system, in order to possibly improve the process at run-time. In addition to pure performance prediction, the *Advanced Monitoring* module supports the anticipated notification of Service Level Agreement (SLA) violations, whenever a process instance is estimated to fail a given quality requirement, previously established for one of the performance measures associated with the process.

## 5 Experiments

In order to assess the validity and usefulness of our approach, we conducted a series of tests on the logs of two real-life systems: *(i)* a transshipment system, supporting the handling of containers in a maritime hub, and *(ii)* a bug-tracking system. For the sake of readability, Table 1 reports some major features of both scenarios, concisely referred to as *harbor* and *bug*, respectively.

### 5.1 Test setting

Three variants of the `CCD` algorithm have been studied in our test, which differ in the regression method exploited as base learners (for building the predictor of each discovered cluster): `CCD-AVG`, where each cluster predictor just returns the average performance of its associated instance; `CCD-RT`, based on the regression-tree induction algorithm *RepTree* [26]; and `CCD-IBK`, exploiting a $k$-$NN$ procedure available in Weka [17] with the name of `IBK`. Notice that `CCD-AVG` is just intended to serve as a baseline, giving some feedback on the effectiveness of the mere co-clustering scheme.

**Table 1.** Summary features of the two application scenarios (including the associated event/trace attributes used in the tests).

| Scenario | #events | #cases | Event Attrib. | Trace Attrib. | Target |
|---|---|---|---|---|---|
| *Harbor* | 21484 | 5336 | `movType, shift, area_from, area_to, vehicleType, block_from, block_to` | `service_in, service_out, imo, line_in, line_out, size, height, vessel_in, containerType, reefer, carrierType_in, carrierType_out, prevCall, nextCall, outOfGauge, prevCountry, nextCountry` | remaining time |
| *Bug* | 8661 | 2283 | `assignee, blocks, component, hardware, priority, product, resolution, os, severity, status` | `comments, votes, severity, QA, classification, component, URL, reporter, keywords, resolution, product, assignee, priority, status, hardware, flags` | remaining steps |

In all cases, a fixed setting was used for the parameters of algorithm `CCD`: $maxIter = 20$ (although, actually, all runs always terminated in fewer iterations), $\gamma = 0$, $\sigma = 1\%$, $maxCl_E = maxCl_T = 50$. The bound on the number of event/traces clusters was just set in order to obtain handier process models and make their computation faster, seeing that this leads to very little accuracy losses (at least for the two major methods `CCD-RT` and `CCD-IBK`), with respect to the default setting $maxCl_E = maxCl_T = \infty$ — further details can be found in the sensitiveness study of Section 5.6.

For the sake of comparison, besides the two basic regression algorithms mentioned above (and denoted by `RT` and `IBK`) playing as baselines, we tested the FSM-based method in [4] (here named `AFSM`), and the `CATP` algorithm of [16] (reusing `AFSM` as base learner), and two variants of the approach in [7], denoted by as `AATP-IBK` and `AATP-RT`, which still combine the base regressors `IBK` and `RepTree`, respectively, in a trace clustering scheme. Notably, none of these competitors feature any automated mechanism for abstracting each log event (into an activity/action symbol), and need to be provided with a user-defined event abstraction function.

In order to test the prediction accuracy, we used three standard error metrics (computed via 10-fold cross validation): *root mean squared error (rmse)*, *mean absolute error (mae)*, and *mean absolute percentage error (mape)*. For the sake of statistical significance, all the error results reported in the following have been averaged over 10 trials and computed via 10 fold cross-validation, while also applying a statistical test procedure to check whether methods accuracies are really different. More specifically, for each error metrics, we used a paired two-tail Student's test to compare the outcomes of each method with those of the most precise one (i.e., that achieving the lowest average error), with respect to two different confidence levels: 95% and 99%, In the following, a method will be considered as almost equivalent to (resp., substantially worse than) the best performer if it was deemed as not significantly different at the 95% level (resp., as significantly different at the 99% level) from the best one — i.e. if we could not reject with a 95% level of confidence (resp., we did can reject) the null hypothesis that that method behaves identically to the latter.

## 5.2 Application Scenarios

*Harbor Scenario* This real-life scenario pertains the handling of containers in a maritime terminal, where a series of logistic activities are traced for each all the containers that pass through the harbor (nearly 4 millions per year). In our experimentation, we focused on a subset of 5336 containers (namely those passed through the hub within the third of year 2006, and exchanged with other ports of the Mediterranean sea), while regarding the transit of each container is here seen as a case of a (unknown) logistic process.

As mentioned in Example 1, each event in the log of the system stores, via event attributes, different aspects of the logistics (move) actions performed on a container, which include: *(i)* the source and destination position it was moved between, in terms of yard's blocks (`block_from` and `block_to`, resp.) and areas (`area_from` and `area_to`, resp.) *(ii)* the kind of operation performed (namely **MOV**e, **DR**ive to **B**ring, **DR**rive to **G**et, **LOAD**, **DIS**charge, **SH**u**F**fle, **OUT**), stored by attribute `movType`; *(iii)* the type of instrument used, encoded by attribute `vehicleType`, ranging from cranes to straddle-carriers and multi-trailers.

Trace attributes convey instead different properties of the handled container (seen as a process instance). These include the origin (`prevCountry`) and destination (`nextCountry`) countries, its previous (`prevCall`) and next (`nextCall`) ports, some properties of the ship loading (e.g., `carrierType_in`) it, physical features (e.g., `size` and `height`). Like in [16], 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 time.

A complete list of all events' and traces' attributes can be found in Table 1.

*Bug Scenario* In this scenario we analyzed the Eclipse project's bug repository, developed with *Bugzilla* (*http://www.bugzilla.org*), a general-purpose bug tracking system offering a range of basic services for handling software bugs (e.g., track a bug, exchange messages about it, submit/review patches).

Some major fields associated with each bug $b$ are: who reported the bug (`reporter`), and who it was allocated to (`assignee`); different features of the software module affected (e.g., `component`, `product`, `version`, `hardware`); its `severity` and `priority` levels; the number of comments made on $b$ (`comments`); the lists of other bugs that must wait for $b$ being solved (`blocks`); the `status` and `resolution` of $b$. Almost all bug fields (apart from the `reporter`) may change dynamically as the bug case proceeds. In particular, the status of a bug $b$ can take one of the following values: *unconfirmed*, *new*, *assigned*, *resolved*, *verified*, *reopened*, and *closed*. For a resolved bug $b$, the `resolution` may be: *fixed*, *duplicate*, *works-for-me*, *invalid*, or *won't-fix*.

As to log events, in Bugzilla the history of a bug is stored in terms of update records (named "bug activities"), with five fields: *who* (the person who made the update), *when* (timestamp), *what* (the attribute modified), *removed* (the former value) and *added* (the new value). Changes made to multiple bug fields during a single access session are treated as a single "bug activity". We encoded each bug

**Table 2.** Prediction results on the *harbor* scenario: errors made (over remaining times) by CCD and several competitors. For each metrics, the **best** outcome is reported in bold and underlined, while all methods **nearly equivalent** to the best one, and those *neatly worse* than it (according to T-test) are shown in bold and in italics, respectively.

| **Predictors** | | **Error Measures** | | |
|---|---|---|---|---|
| *Approach* | *Methods* | rmse | mae | mape (%) |
| Algorithm CCD (Fig. 1) | CCD-IBK | **26.57±8.11** | **5.39±8.11** | **15.00±11.34** |
| | CCD-RT | **25.39±8.38** | **5.95±0.91** | **10.17±10.61** |
| | CCD-AVG | *28.58±11.44* | *8.27±1.44* | *38.10±15.86* |
| Competitors with setting *S1* (1-attribute event abstraction) | AATP-IBK [7] | 31.93±12.50 | *7.04±1.20* | *63.62±5.65* |
| | AATP-RT [7] | 29.95±9.67 | *8.76±1.31* | *66.32±14.80* |
| | AFSM [4] | *80.46±11.93* | *30.74±1.40* | *279.15±26.72* |
| | CATP [16] | 31.53±8.33 | *8.35±0.60* | *58.26±26.96* |
| | IBK [26] | *33.66±9.37* | *7.64±0.89* | *72.50±9.85* |
| | RT [26] | *30.28±8.91* | *8.36±0.77* | *69.67±8.70* |
| Competitors with setting *S2* (5-attribute event abstraction) | AATP-IBK [7] | *54.38±7.98* | *16.66±2.00* | *288.55±44.82* |
| | AATP-RT [7] | *43.84±8.08* | *16.58±1.08* | *144.19±46.31* |
| | AFSM [4] | *75.31±16.68* | *25.27±2.77* | *53.95±20.27* |
| | CATP [16] | *56.21±11.68* | *20.25±1.73* | *85.56±34.64* |
| | IBK [26] | *54.02±5.97* | *16.50±1.42* | *290.54±28.76* |
| | RT [26] | *43.33±6.04* | *15.59±0.76* | *205.06±48.89* |

activity *a* into an event having as many attributes as the number of (modifiable) bug fields, where the value of each attribute is either *(i)* the new value assigned to the corresponding field, if it was really modified in *a*, or *(ii) null*, otherwise. Notably, this flat representation allows propositional mining technique to capture the "simultaneous" modification of multiple bug fields.

As a sample for our experiments, we used a subset of the bugs created from January 1st, 2012 to April 1st, 2013, which were all fixed at least once, but not opened and closed in the same day, while also filtering out all events (i.e. bug activities) that did not refer at least one of the fields `status`, `resolution`, and `assignee`. The resulting log was composed of 2283 traces, with lengths (i.e. nr. of events) ranging from 2 to 25.

### 5.3 Prediction Accuracy Results

*Results on the* Harbor *Scenario* Table 2 shows the averages and standard deviations for the errors made by our methods and the competitors/baseline ones, when trying to predict the remaining processing of container traces.

While our approach doesn't need any preliminary event abstraction/labelling, and it can work directly with raw (multi-dimensional) event tuples, all previous approaches need that an event abstraction criterion is specified in advance. Therefore, we considered two different settings for applying them to the harbor scenario: *(S1)* abstracting each container handling events with just the associated move type (namely, `MOV`, `DRB`, `DRG`, `LOAD`, `DIS`, `SHF`, or `OUT`), as stored in attribute `moveType`; and *(S2)* using a combination of more event attributes (namely, `movType`, `shift`, `vehicleType`, `area_from`, `area_to`).

**Table 3.** Prediction results on the *bug* scenario: errors made (over remaining steps) by `CCD` and several competitors. The **best** result is in bold and underlined, methods **nearly equivalent** to the best one are in bold, those *neatly worse* than it in italics.

| Predictors | | Error Measures | | |
|---|---|---|---|---|
| *Approach* | *Methods* | `rmse` | `mae` | `mape (%)` |
| Algorithm `CCD` (Fig. 1) | `CCD-IBK` | **1.369±0.666** | **0.448±0.111** | **0.167±0.010** |
| | `CCD-RT` | **1.345±0.658** | 0.496±0.101 | *0.210±0.008* |
| | `CCD-AVG` | *1.440±0.666* | *0.578±0.118* | 0.197±0.005 |
| Competitors, provided with ad-hoc defined activity labels | `AATP-IBK` [7] | **1.369±0.446** | 0.472±0.143 | **0.176±0.020** |
| | `AATP-RT` [7] | 1.381±0.723 | *0.566±0.128* | *0.767±0.134* |
| | `AFSM` [4] | *1.463±0.818* | *0.590±0.164* | *0.779±0.035* |
| | `CATP` [16] | *1.404±0.839* | *0.578±0.175* | *0.684±0.041* |
| | `IBK` [26] | 1.392±0.848 | 0.484±0.164 | *0.555±0.041* |
| | `RT` [26] | *1.499±0.787* | *0.637±0.154* | *0.873±0.020* |

Clearly, when faced with the challenge of dealing with complex events, according to setting *S2*, all the competitors show a neat worsening of results[1], with respect to the case where they were just made focus on the kinds of moves performed (setting *S1*). In fact, the latter setting corresponds to a natural choice for traditional (workflow-oriented) process mining approaches, since the performed move operation is indeed the closer than any other event attributes to the notion of process task. Interestingly, we verified empirically that this is the best single-attribute event abstraction — i.e. worse results were obtained by existing methods when the events were preliminary abstracted with the value of another attribute of them. By a finer grain analysis, it easy to see that, generally, all base predictors `IBK`, `RT` and `AFSM` get improved when integrated in a context-driven trace clustering scheme (see `AAPT-IB`, `AAPT-RT` and `CATP`, respectively) – actually, this effect is very marked in the last case.

However, the best outcomes are right discovered by our methods `CCD-IBK` and `CCD-RT` — remember that `CCD-AVG` is just given as a baseline, measuring co-clustering loss. Besides confirming the ability of algorithm `CCD` to automatically find an abstract representations of raw events (hinged on the features of them that are really relevant to performance prediction), these results show the superiority of our approach with respect to the two-phase (i.e. event abstraction, followed by model induction) strategy commonly followed in the field of process mining, as well as to the very idea of using simple event abstraction methods, centered on just one event property, or on a combination of a fixed number of them.

*Results on the* Bug *Scenario* The prediction errors measured for all approaches against the *bug* scenario are reported in Table 3. In order to provide the competitors with a suitable definition of abstract events, we tried different combinations of bug fields as possible activity labels, and empirically found that the best prediction results are achieved (by competitors) when focusing only on the changes made to a bug's `status` (and to the `resolution` field, if modified "contempo-

[1] Notice that setting *S1* was right focused on a subset of all event attributes available, in order not to penalize excessively the competitors — yet considering unnatural for an analyst to define abstract activities based on many event properties.

**Table 4.** Some event clusters (left) and trace clusters (rigth) found with `CCD-RT` on the *harbor* scenario, with $\sigma = 0.01$, and $maxCl_E = maxCl_T = 50$.

| id | condition | size |
|---|---|---|
| $\hat{e}_1$ | `area_to` $\in \{C, BFS, SR\} \wedge$ `area_from` $\in \{A\text{-}NEW, T, B\text{-}NEW, \ldots\}$ | 12% |
| $\hat{e}_2$ | `area_to` $\in \{C, BFS, SR\} \wedge$ `area_from` $\in \{C, CR, A, \ldots\}$ | 5% |
| $\hat{e}_3$ | `area_to` $\in \{CR, BITTE\}$ | 17% |
| $\hat{e}_4$ | `area_to` $\in \{MTR, T, GT, \ldots\}$ | 13% |
| $\hat{e}_9$ | `area_to` $\in \{A\text{-}NEW, A, B\text{-}NEW, \ldots\} \wedge$ `movType` $= MOV \wedge$ `area_from` $\in \{A\text{-}NEW, A, BITTE, \ldots\}$ | 11% |
| $\hat{e}_{12}$ | `area_to` $\in \{A\text{-}NEW, A, B\text{-}NEW, \ldots\} \wedge$ `movType` $\in \{OUT, DRG, LOAD, DRB, DIS, SHF\} \wedge$ `area_from` $\in \{A\text{-}NEW, A, BITTE, \ldots\}$ | 6% |

| id | condition | size |
|---|---|---|
| $\hat{t}_{20}$ | $count(\hat{e}_1) \leq 0 \wedge count(\hat{e}_3) \leq 0 \wedge count(\hat{e}_4) \leq 0 \wedge count(\hat{e}_9) > 0 \wedge$ `nextCountry` $\in \{BG, BE, KR, \ldots\} \wedge$ `service_OUT` $\in \{ME3, GBX, \ldots\}$ | 6% |
| $\hat{t}_{33}$ | $count(\hat{e}_1) \leq 0 \wedge count(\hat{e}_2) \leq 0 \wedge count(\hat{e}_3) \leq 0 \wedge count(\hat{e}_4) \leq 0 \wedge count(\hat{e}_9) \leq 0 \wedge count(\hat{e}_{12}) \leq 0 \wedge$ `nextCountry` $\in \{GE, HR, TN, \ldots\} \wedge$ `service_OUT` $\in \{AEC, GAX, \ldots\} \wedge$ `line_OUT` $\in \{CPP, CPS, SEN, \ldots\}$ | 4% |
| $\hat{t}_{44}$ | $count(\hat{e}_1) \leq 0 \wedge count(\hat{e}_3) \leq 0 \wedge count(\hat{e}_4) \leq 0 \wedge count(\hat{e}_9) \leq 0 \wedge$ `nextCountry` $\in \{GR, ES, AE, \ldots\} \wedge$ `service_OUT` $\in \{EEX, BSS, \ldots\} \wedge$ `line_OUT` $\in \{MSK, APL, HLL, \ldots\} \wedge$ `prevCountry` $\in \{LB, SY, BE, EG, DZ\}$ | 2% |

raneously"), or to its `assignee`. In the former case, the activity label will also encode the newly values assigned (to `status` and `resolution`), whereas no details are kept about the individual to whom the bug was (re-)assigned — as this would lead to overfitting. The resulting abstract events took looked like the following activity labels: `status`:=new, `status`:=resolved + `resolution`:=fixed, etc., $\Delta$`assignee` — with the latter encoding any change to the `assignee` field.

Despite the fact that it was not provided with any suggestion on how events should be abstracted, our approach reached excellent prediction results (except when using the naïve regressor `CCD-AVG`), neatly better than all competitors but `AATP` (and, partially, the basic `IBK` method).

## 5.4 Qualitative Results

This section is meant to provide the reader with some example of the event/trace classification functions, which actually underly our performance prediction models, and are discovered by our approach in an automated unsupervised manner.

*Models found on the* Harbor *Scenario* Table 4 summarizes the classification rules associated with some of the clusters discovered by our approach (namely, by the `CCD-RT`) out of the harbor log. Clearly, these rules are quite easy to interpret and validate, and practically demonstrate the capability of our approach to provide the analyst with useful descriptions of process behavior, in addition to guarantee accurate predictions. In particular, the event clusters shown in the table confirm that performance-relevant activity patterns cannot be captured by just one of the event properties, nor by a fixed combination of them. Interestingly, indeed, while some event clusters just correspond to a subset of destination areas, other of them also depend on the destination area, or even further on the kind of move performed. On the other hand, the descriptions of trace clusters let us recognize the presence of different execution scenarios for the process, which depends both on context factors (e.g., the country of the previous/next port, or the line/service

**Table 5.** All event clusters (left) and some trace clusters (right) found by algorithm $CCD$ on the *bug* scenario, with $\sigma = 0.01$, and $maxCl_E = maxCl_T = 50$.

| id | condition | size |
|---|---|---|
| $\hat{e}_1$ | `status` $=$ *closed* | 47% |
| $\hat{e}_2$ | `status` $=$ *verified* | 2% |
| $\hat{e}_3$ | `status` $\in \{resolved, new\} \wedge$ `resolution` $=$ *fixed* | 38% |
| $\hat{e}_4$ | `status` $\in \{resolved, new\} \wedge$ `resolution` $\in \{worksforme, invalid\}$ | 1% |
| $\hat{e}_5$ | `status` $=$ *assigned* | 8% |
| $\hat{e}_6$ | `status` $=$ *reopened* | 4% |

| id | condition | size |
|---|---|---|
| $\hat{t}_1$ | $count(\hat{e}_1) > 0 \wedge$ `comments` $> 6 \wedge$ `component` $\in \{build, foundation, \ldots\}$ | 1% |
| $\hat{t}_{22}$ | $count(\hat{e}_1) \leq 0 \wedge 5 <$ `comments` $\leq 15 \wedge$ $count(\hat{e}_2) \leq 0 \wedge count(\hat{e}_3) > 0$ `component` $\in \{DBWS, Graphiti, \ldots\}$ | 4% |
| $\hat{t}_{47}$ | $count(\hat{e}_1) \leq 0 \wedge count(\hat{e}_3) \leq 0 \wedge$ $count(\hat{e}_5) \leq 0 \wedge$ `comments` $\leq 10 \wedge$ `product` $\in \{Xtend, Aether, Jetty, \ldots\} \wedge$ `component` $\in \{Xpand, Debugger, \ldots\}$ | 3% |

that was planned to bring the container) and on some of the discovered event clusters (here playing the role of high-level activity patterns).

*Models found on the* Bug *Scenario* The qualitative results in Table 5 confirm that our approach did manage to automatically extract a suitable abstract representations for the given log events, which looks indeed very similar to the optimal one defined by expert for the application of competitors, while recognizing different process' usage scenarios (i.e. trace clusters) depending on the discovered event classes. Indeed, the `status` and `resolution` attributes have been fully exploited for discriminating among event classes. Trace clusters (capturing different execution scenarios) seems to depend mainly on the number of comments associated with bugs, and on the component and/or product affected — as well as on the occurrences of some of the discovered event classes (capturing high-level activity patterns).

## 5.5 Post-processing Results

Figure 3 finally shows a data-driven aggregation hierarchy discovered for the trace attribute `nextCountry` (i.e. the country of the next port that a container is planned to reach), which was computed automatically on the basis of a contingency matrix derived by our system (module `OLAP gateway`), according to the distribution of this attribute's values across all discovered trace clusters. Clearly, this capability is particularly interesting for attributes with high cardinality, in that it allows the analyst to have a summarized vision of the attribute's values, which can be taken into account when a new datacube for the analysis of process performances is to be developed.

Figure 4 illustrates instead some aggregated statistics for the distribution of pre-defined user groups (actually corresponding frequent email domains), computed by the same system module for the *bug* scenario.

## 5.6 Sensitiveness to Parameters

In order to analyze how the behavior of algorithm `CCD` depends on its input parameters, we carried out a further series of tests, over both application scenarios

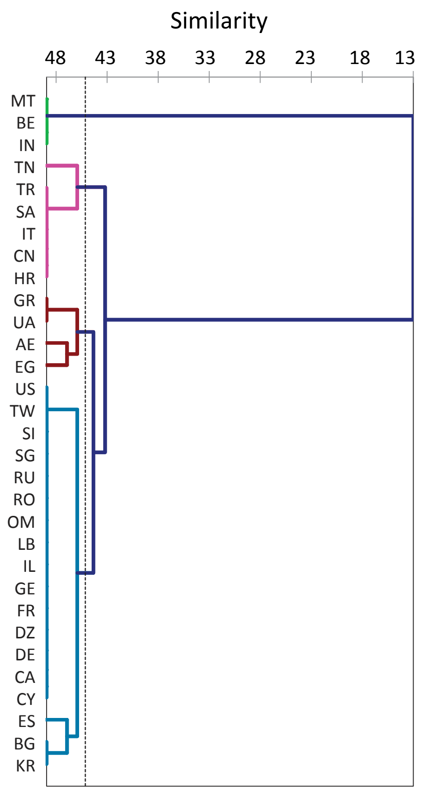


**Fig. 3.** Different aggregations for event attribute `nextCountry`, based on contingency analysis of discovered event clusters (*harbor* scenario).

described above. Figure 5 reports the three kinds of error metrics obtained on the *harbor* scenario by the three instantiations of the algorithm (namely, `CCD-RT`, `CCD-IBK`, and `CCD-AVG`), when varying the setting of parameters. More precisely, a distinct curve is depicted for each version of `CCD`, for different values of the maximum numbers of event/trace clusters (namely, $maxCl_E \in \{25, 50, INF\}$, $maxCl_T \in \{50, 100, INF\}$) and of the clusters' coverage threshold (namely, $\sigma = \{0, 0.01\}$). Notice that label "INF" stands here for the case where no actual bound is set on the number of clusters, which will be decided autonomously by the PCT learning algorithm (based on its default stopping and pruning criteria).

Clearly, the regression method used as base PCM learner is the factor exhibiting the strongest impact on precision results. In particular, the disadvantage of using the naïve regressor (`CCD-AVG`) is neat, especially when a low threshold for $\sigma$ is used — and hence bigger clusters are found, likely exhibiting some higher level of variability in the distribution of the target metrics. The other two methods seem to achieve satisfactory accuracy (especially when using a (quite small) lower bound for clusters' coverage), with `CCD-RT` performing slightly bet-

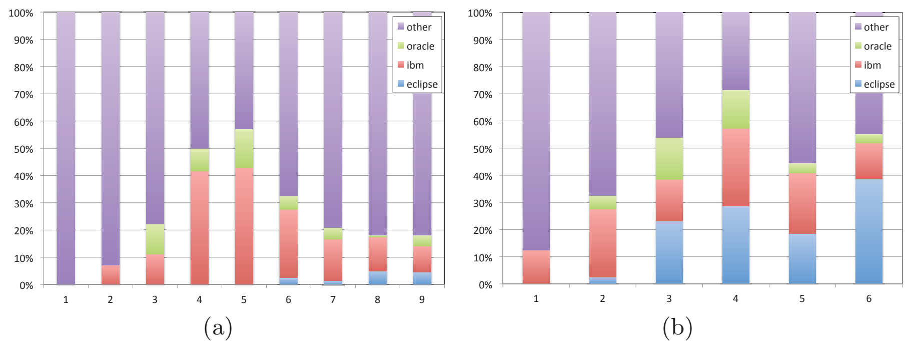


**Fig. 4.** Distribution of `reporter` (a) and `assignee` (b) groups in some of the discovered trace clusters for the *bug* scenario.

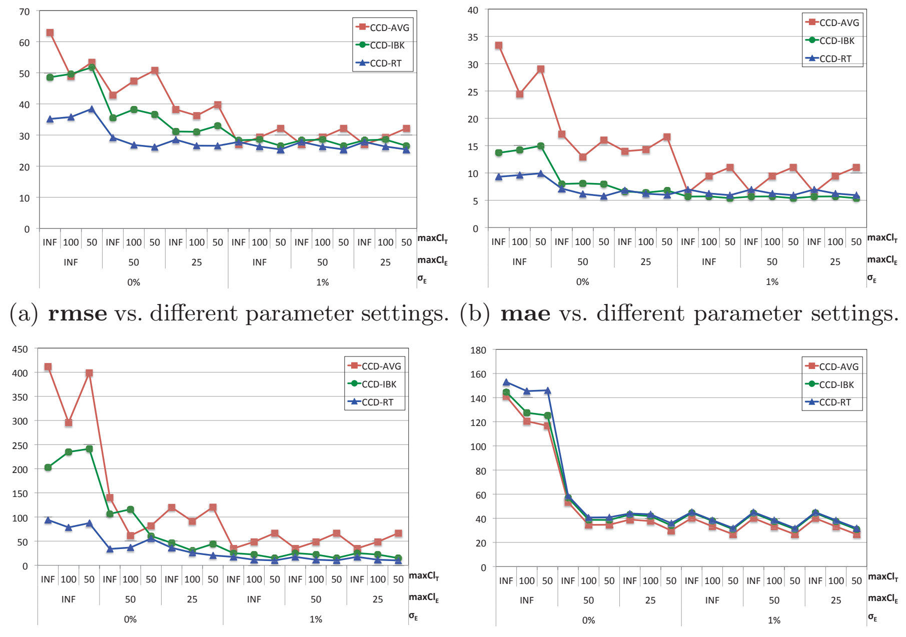
ter than `CCD-IBK`, irrespective of other parameters. This is a good news, since the `RepTree` regression method is to be preferred to `IBK` for both readability and scalability reasons.

As the to remaining parameters, it is easily seen that, no matter of the error metrics, poorest results are obtained when $\sigma = 0$, i.e. no preventive filtering is applied to lowly frequent clusters. The call for a frequency cut on, especially on event clusters, is strengthen by the observation that when acting directly on the maximum number of event clusters (e.g., $maxCl_E = 50$ or $maxCl_E = 25$), we still obtain good results, even when $\sigma = 0$.

Anyway, our approach seems to enjoy good stability and robustness, over a wide range of parameter configurations. Indeed, as soon as a sufficient coverage level for the cluster is fixed (e.g., $\sigma = 1\%$), the remaining two parameters, $maxCl_E$ and $maxCl_T$, do not impact on the quality of predictions anymore. The only exception is `CCD-AVG`, which, as mentioned before, is negatively biased by the reduction of trace clusters. This behavior can be easily explained by observing that if $maxCl_T$ drops, different usage scenarios tend to be mixed into the trace clusters, and the trivial average predictor (which simply returns the means of the performance values in a cluster), tends to make higher prediction errors. Conversely, the other methods seem to cope well with the presence of heterogenous behaviors in the same cluster, thanks to their capability to build a more refined prediction model for the cluster.

In practice, it seems sufficient to choose an adequate value of $\sigma$, in order to ensure good and stable prediction outcomes, no matter of the remaining parameters – which might be, indeed, harder to tune in general.

With the help of Figure 5.(d), reporting (in seconds) the computation times of all our methods, we can also conclude that using a suitable setting for $\sigma$ dramatically improves the scalability of our approach. A minor speed-up effect is played instead by the $maxCl_T$ parameter, which actually reduces the number of training instances that are to be processed by the base learners, at the end of co-clustering procedure.



(a) **rmse** vs. different parameter settings. (b) **mae** vs. different parameter settings.

(c) **mape** vs. different parameter settings. (d) Scalability trends.

**Fig. 5.** Effect of some major parameters of algorithm CCD on the prediction error, w.r.t. different error metrics (tests performed on the *harbor* scenario).

## 6 Discussion and Conclusions

The method proposed in the paper enhances current process mining approaches for the analysis of business process performances in different respects. First of all, it allows for removing the common assumption that the traced event logs explicitly refer (or can be easily mapped to) some well defined process tasks, and automatically replaces (in data-driven way) the formers with high-level activity patterns, expressed at the right level of abstraction for the characterization of performance behaviors. By inducing a definition of both abstract activities of process variants, the approach suits well the case where low level log data are available for analyzing the behavior of a complex processes, exhibiting diverse context-dependent variants. This frees the analyst from the burden of explicitly defining a mapping between log records and high level process tasks, possibly using some event abstraction tool for preprocessing the log. The empirical analysis conducted on two different application scenarios, showed that the approach manages to achieve compelling prediction accuracy with respect to state-of-the-art methods, developed in the field of Process Mining, even when these latter are provided with an expert-driven definition of process activities (i.e. with carefully chosen event abstraction rules). Notice that our empirical analysis was only meant to demonstrate the ability of our approach to find a collection of good quality predictors (fitting different unknown context-dependent execution scenarios), by automatically reaching a right level of abstraction over lowly structured multidimensional log records. In fact, more powerful, regression algorithms can be integrated in our approach, which might have achieved even better performances than the basic ones mentioned above. In a sense, we are mainly interested in providing a proof-of-concept for the innovative idea of supporting the analysis of process performances through the combination of context-based inductive methods for event abstraction and trace clustering.

Moreover, the very classification functions discovered for both events and traces (expressed in terms of logical rules over event/case attributes, and discovered in unsupervised way, via an ad hoc predictive co-clustering scheme) give an important descriptive value to the model, which can turn it very useful for the analyst to fully comprehend the behavior of the process, and the dependence of its performances on both context factors and activity patterns — besides allowing for a quick validation and evaluation of the the discovered models. In fact, feature distinguish our approach from most of those presented so far in the literature for clustering either log traces or log events, where (with the exception of [16, 7]) the discovered trace/activity clusters are not self-explicative, and need further efforts for being interpreted, and validated.

Finally, as to efficiency issues, despite being based on multiple applications of the basic predictive clustering procedure to the input log, when applying our approach, with $\sigma = 1\%$ and $maxCl_E = maxCl_T = 50$, to the real scenarios presented above, each computation only took about 3.6 times the computation time of the fastest among the competitors — excluding IBK, which only keeps a copy of the input traces in main memory, without performing any real learning task. This worst case ratio goes up to 5.4 when no finite upper bound is set for the numbers of clusters. Considering the benefits that our approach can bring (in terms of accuracy, usability, and interestingness of the knowledge discovered), these extra costs look well acceptable.

As to future work, we plan to investigate on enhacing the expressive power of our event/trace classification models, by possibly resorting to First-Order Logic induction techniques, as well as to integrate advanced regression methods for learning cluster predictors, and to complete and extend the OLAP capabilities of our prototype system, Moreover, we would like to extend the approach to the discovery of control-flow process models, by suitably redefining the objective function in our core co-clustering scheme, the encoding of both traces and events, and obviously the kind of base learner for inducing the model of each cluster.

## References

1. CLUS: A predictive clustering system. *http://dtai.cs.kuleuven.be/clus/*
2. A, P., V, S., M, W.: The triconnected abstraction of process models. In: Proc. 7th Int. Conf. on Business Process Management. pp. 229–244. (BPM'09) (2009)
3. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow mining: a survey of issues and approaches. Data & Knowledge Engineering 47(2), 237–267 (2003)

4. van der Aalst, W.M.P., Schonenberg, M.H., Song, M.: Time prediction based on process mining. Information Systems 36(2), 450–475 (2011)
5. Baier, T., Mendling, J.: Bridging abstraction layers in process mining by automated matching of events and activities. In: Proc. of 11th Int. Conf. on Business Process Management (BPM'13). pp. 17–32 (2013)
6. Banerjee, A., Dhillon, I., Ghosh, J., Merugu, S., Modha, D.S.: A generalized maximum entropy approach to Bregman co-clustering and matrix approximation. J. Mach. Learn. Res. 8, 1919–1986 (2007)
7. Bevacqua, A., Carnuccio, M., Folino, F., Guarascio, M., Pontieri, L.: A data-driven prediction framework for analyzing and monitoring business process performances. In: Enterprise Information Systems, 15th Int. Conf., ICEIS 2013, Revised Selected Papers. To Appear. (2013)
8. Blockeel, H., Raedt, L.D.: Top-down induction of first-order logical decision trees. Artificial Intelligence 101(1-2), 285–297 (1998)
9. Blockeel, H., Raedt, L.D., Ramon, J.: Top-down induction of clustering trees. In: Proc. of the 15th Int. Conf. on Machine Learning (ICML'98). pp. 55–63 (1998)
10. Bose, R.P.J.C., van der Aalst, W.M.P.: Abstractions in process mining: A taxonomy of patterns. In: Proc. of 7th Int. Conf. on Business Process Management (BPM'09). pp. 159–175 (2009)
11. Conforti, R., Fortino, G., Rosa, M.L., ter Hofstede, A.H.M.: History-aware, real-time risk detection in business processes. In: Proc. of 19th Int. Conf. on Cooperative Information Systems (CoopIS'11). pp. 100–118 (2011)
12. Dongen, B.F., Crooy, R.A., van der Aalst, W.M.P.: Cycle time prediction: When will this case finally be finished? In: Proc. of 16th Int. Conf. on Cooperative Information Systems (CoopIS'08). pp. 319–336 (2008)
13. DR, L., M., S.: Workflow modeling for virtual processes: an order-preserving process-view approach. Information Systems 28, 505–532 (2003)
14. Ekanayake, C.C., Dumas, M., García-Bañuelos, L., La Rosa, M.: Slice, mine and dice: complexity-aware automated discovery of business process models. In: Proc. of 11th Int. Conf. on Business Process Management (BPM'13). pp. 49–64 (2013)
15. Folino, F., Greco, G., Guzzo, A., Pontieri, L.: Mining usage scenarios in business processes: Outlier-aware discovery and run-time prediction. Data & Knowledge Engineering 70(12), 1005–1029 (2011)
16. Folino, F., Guarascio, M., Pontieri, L.: Discovering context-aware models for predicting business process performances. In: Proc. of 20th Int. Conf. on Cooperative Information Systems (CoopIS'12). pp. 287–304 (2012)
17. Frank, E., Hall, M.A., Holmes, G., Kirkby, R., Pfahringer, B.: Weka - a machine learning workbench for data mining. In: The Data Mining and Knowledge Discovery Handbook, pp. 1305–1314. Springer (2005)
18. Garofalakis, M., Hyun, D., Rastogi, R., Shim, K.: Building decision trees with constraints. Data Mining & Knowledge Discovery 7(2), 187–214 (2003)
19. Greco, G., Guzzo, A., Pontieri, L.: Mining taxonomies of process models. Data & Knowledge Engineering 67(1), 74–102 (2008)
20. Greco, G., Guzzo, A., Pontieri, L., Saccà, D.: Discovering expressive process models by clustering log traces. IEEE Trans. on Knowl. and Data Engineering 18(8), 1010–1027 (2006)
21. Günther, C.W., Rozinat, A., van der Aalst, W.M.P.: Activity mining by global trace segmentation. In: Proc. of Business Process Management Workshops (BPI'09). pp. 128–139 (2009)

22. Milani, F., Dumas, M., Matulevičius, R.: Decomposition driven consolidation of process models. In: Proc. of 25th Int. Conf. on Advanced Information Systems Engineering (CAiSE'13). pp. 193–207 (2013)
23. Schonenberg, H., Weber, B., Dongen, B., van der Aalst, W.P.M.: Supporting flexible processes through recommendations based on history. In: Proc. of the 6th 10th Int. Conf. on Business Process Management (BPM'08). pp. 51–66 (2008)
24. Song, M., Günther, C.W., van der Aalst, W.: Trace clustering in process mining. In: Proc. of Business Process Management Workshops (BPI'08). pp. 109–120 (2008)
25. Vara, J.L.D.L., Ali, R., Dalpiaz, F., Sánchez, J., Giorgini, P.: COMPRO: a methodological approach for business process contextualisation. In: Proc. of 18th Int. Conf. on Cooperative Information Systems (CoopIS'10). pp. 132–149 (2010)
26. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems). Morgan Kaufmann Publishers Inc. (2005)