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

Mining multi-variant process models from low-level logs

Francesco Folino, Massimo
Guarascio, Luigi Pontieri

RT-ICAR-CS-14-05 **Dicembre 2014**

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*

Mining multi-variant process models from low-level logs

Francesco Folino, Massimo Guarascio and Luigi Pontieri

ICAR-CNR, National Research Council, via P. Bucci 41C, 87036, Rende, CS, Italy,
{ffolino,guarascio,pontieri}@icar.cnr.it

Abstract. Process discovery techniques are a precious tool for analyzing the real behavior of a business process. However, their direct application to lowly structured logs may yield unreadable and inaccurate models. Current solutions rely on event abstraction or trace clustering, and assume that log events refer to well-defined (possibly low-level) process tasks. This reduces their suitability for logs of real BPM systems (e.g. issue management) where each event just stores several data fields, none of which fully captures the semantics of performed activities. We here propose an automated method for discovering an expressive kind of process model, consisting of three parts: *(i)* a logical event clustering model, for abstracting low-level events into classes; *(ii)* a logical trace clustering model, for discriminating among process variants; and *(iii)* a set of workflow schemas, each describing one variant in terms of the discovered event clusters. Experiments on a real-life data confirmed the capability of the approach to discover readable high-quality process models.

Key words: Business Process Mining, Log Abstraction, Trace Clustering.

1 Introduction

Workflow discovery techniques [1] have gained attention in BPM applications, owing to their ability to extract (out of historical execution data) a descriptive model for the behavior of a process, which can support key process analysis/design, process improvement and strategic decision making tasks.

However, two critical issues undermine the effectiveness of traditional workflow discovery methods, when they are applied to the logs of lowly-structured processes: *(i)* the high level of details that usually characterizes log events, which makes it difficult to provide the analyst with an easily interpretable description of the process in terms of relevant business activities, and *(ii)* the presence of various execution scenarios (a.k.a. "process variants"), which exhibit different business processing logics (often determined by key context factors), and cannot be captured effectively with a single workflow model. In fact, when applied to such logs, most current workflow discovery techniques tend to yield "spaghetti-like" models, suffering from both low readability and low fitness [7].

Two kinds of solution methods have been proposed in the literature to alleviate these problems: *(i)* turn raw events into high-level activities by way of automated

abstraction techniques [10, 6], *(ii)* partition the log into trace clusters [8, 5, 4, 9], capturing homogenous execution groups, and then separatley model each cluster with a simpler and more fitting workflow. Since both kinds of methods assume that each log event refers to a predefined (possibly low-level) process task, they are of limited usefulness for many real-world flexible BPM applications (e.g., product management, or problem/issue tracking). In such cases, indeed, each log event takes the form of a tuple storing several data fields, none of which can be interpreted as a task label (capable to fully capture the semantics of the performed activity). Moreover, the clusters produced by current event abstraction (resp., trace partitioning) approaches are not self-descriptive, and the analyst must carry out difficult and long interpretation/validation tasks to turn them into meaningful classes of activities (resp., process instances).

Contribution In order to overcome these limitations, a two-fold mining problem is stated in this work, for a given low-level multi-dimensional event log. On the one hand, we want to discover an event clustering function, allowing for automatically abstracting log events into (non a-priori known) event types, each of which is meant to represent a distinguished pattern for the execution of single work items. While exploiting such event abstraction, we also want to possibly detect and model different process execution variants.

Our solution approach relies, first of all, on a specialized logic-based event clustering method, which can find a clustering function encoded in terms of decision rules (over event attributes), to provide the analyst with an interpretable description of the discovered activity types (i.e. event clusters). We also find a similar conceptual clustering function for partitioning the log traces into different execution classes, by using their associated context data (e.g. cases' properties or environmental factors) as descriptive variables. To this end, an iterative trace-clustering approach is proposed, which tries to greedily maximize the (average) quality of the workflow schemas induced from the discovered trace clusters.

Expressing trace/event clusters via predictive (logical) rules is an important distinguishing feature of our approach, which makes it possibly support the implementation of advanced run-time services. This point is discussed in Section 6, which also provides a comparison with related work in the literature.

2 Preliminaries

Log data For each process case a *trace* is recorded, 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. For each trace $\tau \in \mathcal{T}$, $len(\tau)$ denotes the number of events stored in τ, while $\tau[i]$ is i-th event in τ, for $i \in \{1, ..., len(\tau)\}$.

For each event $e \in E$, let $prop(e)$ be a tuple data properties (over some attribute space) associated with e —each event also refers to a case ID and to a timestamp, but these are useless for recognizing general activity patterns. Execution cases as well are often associated with a number of data properties. For

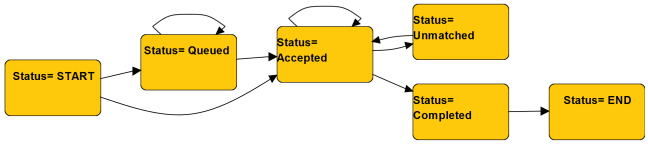

Fig. 1. Workflow schema induced from the *Problem Management* log, when using a `status`-based event abstraction.

each trace $\tau \in \mathcal{T}$, let $prop(\tau)$ be the data properties of τ. Finally, let $traces(L)$ (resp., $events(L)$) denote the set of traces (resp., events) stored in L.

Workflow Schemas Our final aim is to describe process behavior by way of flow-oriented models, like those commonly used to express the control-flow logics of a business process. For the sake of concreteness, we here focus on the language of *heuristics nets* [14], where a *workflow schema* is essentially a directed graph, where each node represent a process activity, and each edge (x, y) encodes a dependency of y on x. In addition, one can express cardinality-based fork (resp., join) constraints for nodes with multiple outgoing (resp., incoming) edges.

For any workflow schema W, let $\mathcal{A}(W)$ be the set of all activities featuring in W. The definition of $\mathcal{A}(W)$ is a crucial point in our setting, where it may well happen that no predefined business activities exist for the process. In such a case, our approach consists in partitioning log events into event classes, representing distinguished activity patterns, which can be eventually used to label the nodes of a workflow schema.

Example 1. Let us consider a real-life Problem Management application (used as a case study in Section 5), where each log trace registers the history of a ticket. Each log event e is a tuple featuring 8 data attributes: the `status` (*accepted*, *queued*, *completed*, or *closed*) and `substatus` (*unmatched*, *awaiting_assignment*, *assigned*, *in_progress*, *wait*, *cancelled*, or *closed*) of the ticket when e happened; the `resource` who generated e, and her nationality (`res_country`); the `support team`, `functional division` and `organization line` which the resource was affiliated to; the country where the line was located (`org_country`). Since such representation carries no information on what process task was performed in each event, we need to infer activity types from event tuples, to grasp some suitable abstraction level on process behavior. A common approach consists in abstracting each event tuple into just one of its attributes, and interpret it as an activity label. Figure 1 shows the model discovered (by algorithm Heuristics Miner [14]) after replacing each event with its respective `status`'s value. ◁

Figure 1 evidences that a one-attribute event abstraction is likely to produce a lowly informative (or even trivial) process model for a log storing fine-grain execution traces. On the other hand, using a combination of multiple attributes to the same purpose may well yield a cumbersome and overfitting workflow schema, as confirmed by our tests. The latter solution is, in fact, also unviable in many real application scenarios for scalability reasons, seeing as the computation

time of typical approaches (to both workflow discovery and log abstraction) is quadratic in the number of activities.

Behavioural Profiles *Behavioural profiles* [13] are basic ordering relationships over activity pairs, which can summarize a workflow schema's behavior, and can be computed efficiently for many classes of workflow specification languages.

Let W be a workflow schema, and $\mathcal{A}(W)$ be its associated activities. Let $\succ^W$ be a "weak order" relation inferred from W, such that, for any $x, y \in \mathcal{A}(W)$, $y \succ^W x$ iff there is at least a trace admitted by W where y occurs after x. Then, the *behavioral profile matrix* of W, denoted by $\mathcal{B}(W)$, is a function mapping each pair $(x, y) \in \mathcal{A}(W) \times \mathcal{A}(W)$ to an ordering relation in $\{\rightsquigarrow, +, \|\}$, as follows: **(i)** $\mathcal{B}(W)[x, y] = \rightsquigarrow$, iff $y \succ^W x$ and $x \not\succ^W y$ (*strict order*); **(ii)** $\mathcal{B}(W)[x, y] = +$, iff $x \not\succ^W y$ and $y \not\succ^W x$ (*exclusiveness*); **(iii)** $\mathcal{B}(W)[x, y] = \|$, iff $x \succ^W y$ and $y \succ^W x$ (*observation concurrency*).

Let τ be a trace in $\mathcal{T}$ (with event universe E), x and y be two event classes (i.e. disjoint subsets of E), and $\mathcal{B}$ be a behavioral profile matrix. Then we say that τ *violates* (resp., *satisfies*) $\mathcal{B}[x, y]$, denoted by $\tau \nvdash \mathcal{B}[x, y]$ (resp., $\tau \vdash \mathcal{B}[x, y]$), if the occurrences of x and y in τ infringe (resp., fullfill) the ordering constraints in $\mathcal{B}[x, y]$. Precisely, it is $\tau \nvdash \mathcal{B}[x, y]$ iff there exist $i, j \in \{1, ..., len(\tau)\}$ such that $\tau[i] = y$, $\tau[j] = x$, and: either *(i)* $\mathcal{B}[x, y] = +$, or *(ii)* $\mathcal{B}[x, y] = \rightsquigarrow$ and $i < j$.

Behavioral profiles help us measure workflow conformance, as in what follows.

Definition 1. *Let L be an event log, W be a workflow schema, and $\sigma \in \mathbb{N}$ be a (lower) noise threshold. Then the compliance of W w.r.t. L, denoted by $compl(W, L)$, is defined as: $compl(W, L) = \frac{1}{|\mathcal{A}(W)|^2} \times |\{(x, y) \in \mathcal{A}(W) \times \mathcal{A}(W) \mid \neg(\exists^{\geq\sigma}\ \tau \in traces(L)\ s.t.\ \tau \nvdash \mathcal{B}(W)[x, y])\}|$. Moreover, the precision of W w.r.t. L, denoted by $prec(W, L)$, is defined as: $prec(W, L) = \frac{1}{|\mathcal{A}(W)|^2} \times |\{(x, y) \in \mathcal{A}(W) \times \mathcal{A}(W) \mid \mathcal{B}(W)[x, y] = \| \text{ and } \neg(\exists^{\geq\sigma}\tau \in L\ s.t.\ |\{x, y\} \cap tasks(\tau)| = 1)\}|$ where $\exists^{\geq\sigma}$ is a counting quantifier, asserting the existence of at least σ elements in a given set.*[1] □

3 Problem Statement

As discussed above, we want to discover two interrelated clustering models: one for log events, and another for log traces (intended to recognize process variants).

For the sake of interpretability, in both cases we seek a clustering model that can be encoded by decision rules. Let us assume that each rule is a conjunctive boolean formula of the form $(A_1 \in V_1) \wedge (A_2 \in V_2) \wedge \ldots \wedge (A_k \in V_k)$, where, for each $i \in \{1, \ldots, k\}$, A_i is a descriptive attribute defined on some given set Z of data instances, and V_i is a subset of A_i's domain. For any $I \subseteq Z$ and for any such a rule r, let $cov(r, I)$ denote the set of all I's instances that satisfy r.

A *conceptual clustering model* for Z is a list $\mathcal{C} = \langle r_1, ..., r_n \rangle$ of conceptual clustering rules (for some positive integer number n), which defines a partitioning of Z into n parts $P_1, \ldots, P_n$, where $P_i = cov(r_i, Z) / \bigcup_{j=1}^{i-1} cov(r_j, Z)$.

[1] In the tests described in Section 5 we always set $\sigma = 0.01 \cdot |traces(L)|$.

Our ultimate goal is to find a multi-variant process model leveraging two such clustering functions (for events and traces, resp.), as it is formally defined next.

Definition 2. *Let L be a log, and $\mathcal{T}$ (resp., E) the associated trace (resp., event) universe. Then, a* High-Level Process Model *(`HLPM`) for L is a triple $\langle \mathcal{C}^E, \mathcal{C}^{\mathcal{T}}, WS \rangle$ such that:* (i) $\mathcal{C}^E = \langle r_1^E, \ldots, r_p^E \rangle$ *is a conceptual clustering model for E, where $p \in \mathbb{N}$ is the number of event clusters, and r_i^E (for $i = 1, .., p$) is the clustering rule of the i-th event cluster;* (ii) $\mathcal{C}^{\mathcal{T}} = \langle r_1^{\mathcal{T}}, \ldots, r_q^{\mathcal{T}} \rangle$ *is a conceptual clustering model for $\mathcal{T}$, where $q \in \mathbb{N}$ is the number of trace clusters, and $r_j^{\mathcal{T}}$ (for $j = 1, .., q$) is the clustering rule of the j-th trace cluster; $WS = \langle W_1, \ldots, W_q \rangle$ is a list of workflow schemas, where W_k (for k=$1, .., q$) models the k-th trace cluster, using the classes yielded by $\mathcal{C}^E$ as activities.* □

Clearly enough, sub-model $\mathcal{C}^E$ plays as an event abstraction function, which maps each event $e \in E$ to an event class, based on logical clustering rules (expressed on e's properties). Analogously, model $\mathcal{C}^{\mathcal{T}}$ partitions historical execution traces into behaviorally homogenous clusters (via logical clustering rules over traces' properties). Each discovered trace cluster, regarded as a distinct process variant, is also equipped with a workflow schema, where some of the clusters of C^E feature as (high-level) activity nodes.

Our discovery problem can be stated conceptually as the search for an optimal `HLPM` that maximizes some associated quality measure, such as those defined in the previous section.

4 Solution Approach

Technically, we rephrase the discovery of conceptual clustering models (over either events or traces) as a predictive clustering problem. Basically, predictive clustering approaches [3] assume that two kinds of data attributes characterize each element z in a given space $Z = X \times Y$ of instances: *descriptive* attributes and *target* attributes, denoted by $descr(z) \in X$ and $targ(z) \in Y$, respectively. The goal of these approaches is to find a logical partitioning function (of the same nature as our conceptual clustering models) that minimizes $\sum_{C_i} |C_i| \times Var(\{targ(z) \mid z \in C_i\})$, where C_i ranges over current clusters, and $Var(S)$ is the variance of set S.

Different predictive clustering models exists in the literature. Owing to scalability and readability reasons, we preferred *Predictive Clustering Trees* (PCTs), where the clustering function is a (propositional) decision tree.

We next introduce ad-hoc propositional encodings for events and traces, allowing for inducing a clustering model by reusing a PCT learner.

Definition 3. *Let L be a log, over event (resp., trace) universe E (resp., $\mathcal{T}$). Then, the* e-view *of L, denoted by $\mathcal{V}_E(L)$, is a relation containing a tuple $z_{i,j}$ for each $\tau_i \in traces(L)$ and for each $j \in \{1, \ldots, len(\tau_i)\}$ (i.e. for each event $\tau[j] \in events(L)$), such that:* ***(i)*** *$descr(z_{i,j}) = prop(\tau_i[j])$, and* ***(ii)*** *$targ(z_{i,j}) = \langle \frac{j}{len(\tau_i)}, \frac{j}{maxL}, \frac{len(\tau_i)-j}{maxL} \rangle$, where $maxL = \max(\{\ len(\tau) \mid \tau \in traces(L)\})$.* □

In such a log view (acting as training set for predictive clustering), the descriptive (input) attributes each instance $z_{i,j}$ are the data fields in $prop(\tau_i[j])$ (see Sec 2), while the target ones are three indicators (derived from the relative/absolute position j of $\tau_i[j]$ in its surrounding trace). Intuitively, we want events with similar intra-trace positions (and similar properties) to be put together.

For the clustering of traces, we introduce another log view, named *t-view*, where the context data and abstract activities of each trace play as descriptive features, while the target variables express local activity relationships.

Specifically, let $\mathcal{B}$ be a behavioral profile matrix (derived from some workflow schema), and $\alpha : E \rightarrow A$ be an event clustering function. Then, for each trace τ and any activities a_i and a_j in A we define a target variable $v_{\mathcal{B}}(\tau, a_i, a_j)$ as follows: **(i)** $v_{\mathcal{B}}(\tau, a_i, a_j) = 1$ if $\tau \not\models \mathcal{B}[a_i, a_j]$; **(ii)** $v_{\mathcal{B}}(\tau, a_i, a_j) = \frac{f(\tau, a_i, a_j)}{2 \times c(\tau, a_i, a_j)}$ if both a_i and a_j occur in τ, where $c(\tau, a_i, a_j) = |\{(i', j') \mid i', j' \in \{1, \ldots, len(\tau)\} \wedge i' \neq j' \wedge \alpha(\tau[i']) = a_i \wedge \alpha(\tau[j']) = a_j\}|$, and $f(\tau, a_i, a_j) = \texttt{sum}(\{\texttt{sgn}(j' - i') \mid i', j' \in \{1, \ldots, len(\tau)\} \wedge \alpha(\tau[i']) = a_i \wedge \alpha(\tau[j']) = a_j\})$, where `sgn` denotes function *signum*; and **(iii)** $v_{\mathcal{B}}(\tau, a_i, a_j) = \texttt{null}$ if τ does not contain both a_i and a_j. This way, $v_{\mathcal{B}}(\tau, a_i, a_j)$ can capture a violation to a behavioral profile (case i), or keep information on the mutual positions of a_i and a_j, if both occur in τ (case ii).

Definition 4. *Let L be a log over event and trace universes E and $\mathcal{T}$, $A = \{a_1, \ldots, a_k\}$ be a set of event clusters (regarded as abstract activities), and $\alpha : E \rightarrow A$ be an event clustering function. Let also $\mathcal{B} : A \times A \rightarrow \{\rightsquigarrow, +, \|\}$ be a behavioral profile matrix. Then, the* t-view *of L w.r.t. α and $\mathcal{B}$, denoted by $\mathcal{V}_{\mathcal{T}}(L, \alpha, \mathcal{B})$, is a relation containing, for each $\tau \in traces(L)$, a tuple z_τ such that:* ***(i)*** *$descr(z_\tau) = prop(\tau) \oplus act_\alpha(\tau)$, where $\oplus$ denotes tuple concatenation, and $act_\alpha(\tau)$ is a vector in $\{0,1\}^k$ such that, for $i = 1, .., k$, $act_\alpha(\tau)[i] = 1$ iff activity a_i occurs in τ (i.e. iff $\exists j \in \{1, \ldots, len(\tau)\}$ s.t. $\alpha(\tau[j]) = a_i$); and* ***(ii)*** *$targ(z_\tau) = \langle v_{\mathcal{B}}(\tau, a_1, a_1), \ldots, v_{\mathcal{B}}(\tau, a_1, a_k), v_{\mathcal{B}}(\tau, a_2, a_2), ..., v_{\mathcal{B}}(\tau, a_2, a_k), \ldots, v_{\mathcal{B}}(\tau, a_i, a_i), \ldots, v_{\mathcal{B}}(\tau, a_i, a_k), \ldots, v_{\mathcal{B}}(\tau, a_k, a_k)\rangle$.* □

Algorithm `HLPM-mine` Our approach to the discovery of a `HLPM` is illustrated as an algorithm, named `HLPM-mine` and reported in Figure 2. The algorithm follows a two-phase strategy: it first finds a conceptual clustering model (Steps 1-3) for the events, and then computes a collection of trace clusters, along with their associated clustering rules and workflow schemas (Steps 4-24).

Conceptual clustering models are found through function `minePCT`, which leverages a PCT-learning method in [3]. In the algorithm, this function is applied to both an *e-view* (Step 2), and a *t-view* (Step 12). Two further parameters allow to constrain a PCT's growth: the minimal coverage a node must have to be possibly split, and the maximal number of leaves, respectively.

The PCT found for log events is turned into a conceptual clustering model through an iterative bottom-up procedure, named `extractRules`, omitted for lack of space. Basically, starting with the rules of the tree leaves, this procedure replaces each rule r with that of the parent, if $|cov(r, \mathcal{V}_E(L))| < minCov \times |\mathcal{V}_E(L)|$, and the variance of $cov(r, \mathcal{V}_E(L))$ is higher than, or nearly equal to, the

Input: Log L, maximal number $m \in \mathbb{N}$ of trace clusters, minimal clusters' coverage $minCov \in [0,1]$, minimal quality-gain $\gamma \in [0,1]$
Output: An HLPM for $\mathcal{T}$

1. $V := \mathcal{V}_E(L)$; *// compute L's e-view (cf. Def. 3)*
2. $T :=$ `minePCT`$(V, minCov, \infty)$;
3. $C_E :=$ `extractRules`$(T, minCov, \gamma)$; *// C_E is a clustering model for E*
4. Create an initial cluster c_0 gathering all L traces;
5. $w_0 :=$ `mineWF`(L, C_E); compute $\mathcal{B}(w_0)$;
6. $TClust := \{c_0\}$; $W(c_0) := w_0$; $rule(c_0) :=$ [`true`];
7. $Q := \{c_0\}$; *// Q is a queue of candidate trace clusters*
8. **while** $(0 < |TClust| < m)$ and $Q \neq \emptyset$ **do**
9. $c^* = \arg\min_{c \in Q}(|c| \times$ `compl`$(W(c), c))$;
10. **if** $|c^*| \geq minCov \times |traces(L)|$ **then**
11. $V := \mathcal{V}_{\mathcal{T}}(c^*, C_E, \mathcal{B}(W(c^*)))$; *// build a t-view (cf. Def. 4)*
12. $T :=$ `minePCT`$(V, minCov, m - |TClust| + 1)$;
13. Let ΔCl be the clusters associated with T's leaves
14. **for each** cluster c in ΔCl **do**
15. Let $traces(c)$ be the traces assigned to c;
16. $W(c) :=$ `mineWF`$(traces(c), C_E)$; compute $\mathcal{B}(W(c))$;
17. $rule(c) :=$ `mergeRules`$(rule(c^*), rule(c))$;
18. **end for**
19. **if** $\{W(c) \mid c \in \Delta Cl\} \prec^{\gamma} W(c^*)$ **then** *//cf. Def. 5*
20. $TClust := TClust \cup \Delta Cl - \{c^*\}$; $Q := Q \cup \Delta Cl$;
21. **end if**
22. $Q := Q - \{c^*\}$;
23. **end if**
24. **end while**
25. $C_{\mathcal{T}} := \langle\rangle$; $WS := \langle\rangle$; *// inititalize the trace-clustering model and workflow list*
26. **for each** cluster $c \in TClust$ **do**
27. $C_{\mathcal{T}}$.append$(rule(c))$; WS.append$(W(c))$;
28. **end for**
29. **return** $\langle C_E, C_{\mathcal{T}}, WS \rangle$

Fig. 2. Algorithm `HLPM-mine`

parent's variance (precisely, the former is lower than the latter of a fraction γ at most). Notice that, in current implementation this test is performed on a separate pruning set. Whenever a rule r is removed, all the instances in $cov(r, \mathcal{V}_E(L))$ are assigned to the parent rule (i.e. to the rule of the parent of r's node), which is appended to the clustering model, if it does not appear in it yet.

Trace clusters are computed through an iterative partitioning scheme (Step 4-24), where $TClust$ is the set of current trace clusters, and Q just contains the ones that may be further split. For any trace cluster c, $W(c)$ and $rule(c)$ store the associated workflow schema and clustering rule, respectively.

Before the loop, $TClust$ just consists of one cluster, gathering all L's traces. At each iteration, we try to split the cluster c^* in Q that has been given the lowest compliance score by the (profile-based) conformance measure *compl* (see

Def. 1), among all those overcoming the coverage threshold $minCov$. To this end, a PCT model T is induced from the propositional view of cluster c^*, according to the discovered event clustering $\mathcal{C}_E$, and the behavioral profiles of the schema associated with c^* (see Def. 4).

For each leaf cluster c in T, we extract a workflow schema and the associated behavioral profiles, and merge the rule of c with that of the cluster (namely, c^*) T was induced from (Steps 16-17).

At this point, the algorithm verifies if the workflows discovered after splitting cluster c^* really allowed to model more effectively the behavior of c^*'s traces. This check relies on an ad-hoc quality relationship, named γ-*improve*, which is defined next, based on the conformance metrics *compl* and *prec* (see Def. 1).

Definition 5. *Let c be a set of traces, $\{c_1, .., c_k\}$ be a partition of c, W be a workflow schema for c, and* WS $= \{W_1, .., W_k\}$ *be a set of workflow schemas s.t. W_i models c_i and $\mathcal{A}(W_i) \subseteq \mathcal{A}(W)$, for $i = 1, .., k$. Then, for $\gamma \in [0, 1]$, we say that* WS γ-improves W*, denoted by* WS $\prec^{\gamma} W$*, iff* ***(i)*** $\sum_{i=1}^{k} \frac{|traces(c_i)| \cdot compl(W_i, c_i)}{|traces(c)|} > (1 + \gamma) \cdot compl(W, c)$*, or* ***(ii)*** $compl(W, c) \leq \sum_{i=1}^{k} \frac{|traces(L_i)| \cdot compl(W_i, c_i)}{|traces(c)|} \leq (1 + \gamma) \cdot compl(W, c)$ *and* $\sum_{i=1}^{k} \frac{|traces(L_i)| \cdot prec(W_i, c_i)}{|traces(c)|} > (1 + \gamma) \cdot prec(W, c)$. □

We hence assume that the workflows of c^*'s children clusters model the behavior of $traces(c^*)$ better than $W(c^*)$ alone, if they get, in the average, higher compliance (on their respective sub-clusters) than W (on c^* as a whole). When the compliance score keeps unchanged, we still prefer the new schemas if they are more precise than $W(c^*)$.

The split of c^* is eventually kept only if the above improvement relationship holds. In any case, c^* is removed from the list of candidates for further refinement.

Steps 25-28 just build the list of trace clustering rules, and the list of workflow schemas, based on the set of trace clusters eventually left in $TClust$.

5 Experiments

The approach described so far was implemented into a Java prototype system, and tested on the log [2] of a real problem management system, encompassing 1487 traces recorded from January 2006 to May 2012. As explained in Example 1, each log event stores eight data attributes (i.e., `status`, `substatus`, `resource`, `res_country`, `functional_division`, `org_line`, `support_team`, and `org_country`). For each problem case p, two attributes are associated with p's trace: p's `impact` (*medium, low*, or *high*), and the `product` affected by p.

We enriched each trace τ with further context-oriented attributes: *(i)* `firstOrg`, indicating the team associated with τ's first event; *(ii)* `workload`, quantitying the number of problems open on the time, say t_τ, when τ started; *(iii)* several time dimensions (namely, `week-day`, `month` and `year`) derived from t_τ.

In all the tests discussed next, we ran `HLPM-mine` with $minCov = 0.01$, and $\gamma = 0.1$, while using plugin FHM [14] to instantiate function `mineWF`.

[2] Available at `http://www.win.tue.nl/bpi/2013/challenge`

Table 1. Results (avg±stdDev) obtained with the event clustering method (no trace clustering) and manual abstraction criteria. The best value of each column is in bold.

Event Clustering Method	*Fitness*	*BehPrec*	*#nodes*	*#edges*	*#edgesPerNode*
`status`	0.676±0.010	0.543±0.004	**6.0±0.0**	**9.0±0.0**	**3.0±0.0**
`substatus`	0.715±0.057	0.546±0.003	9.0±0.0	18.0±0.0	4.0±0.0
`all`	0.723±0.032	**0.921±0.005**	1464.2±18.5	2224.6±23.9	3.0±0.0
`HLPM-mine(m=1)`	**0.815±0.024**	0.736±0.017	17.0±0.0	32.2±0.8	3.8±0.1

Table 2. Results (avg±stdDev) yielded by different trace clustering methods, after abstracting events with the clusters found by `HLPM-mine(m=1)`. Best scores are in bold.

Trace Clustering Method	*Fitness*	*BehPrec*	*#nodes*	*#edges*	*#edgesPerNode*
`ACTITRAC[8]`	0.690±0.027	**0.796±0.012**	12.6±0.1	20.0±0.4	3.1±0.0
`TRMR[4]`	0.571±0.075	0.751±0.021	10.3±0.2	17.0±0.4	3.3±0.0
`A-TRMR[4]`	0.706±0.015	0.764±0.007	11.6±0.2	18.1±0.4	3.1±0.0
`KGRAM[5]`	0.663±0.023	0.725±0.008	10.7±0.2	16.7±0.4	3.1±0.0
`HLPM-mine(m=`∞`)`	**0.851±0.013**	0.742±0.007	**10.0±0.2**	**13.8±0.3**	**2.6±0.0**

Evaluation Metrics Different kinds of metrics (fitness, precision, generalization) exist for evaluating the quality of a workflow schema. In particular, fitness metrics quantify the capability to replay a given log, and represent the main evaluation criterion [7], while the other metrics serve finer-grain comparisons.

In our tests, we measured the fitness of each discovered heuristics-net according to the *Improved Continuous Semantics Fitness* (named *Fitness* hereinafter) defined in [12]. Basically, the fitness of a schema W w.r.t. a log L (denoted by *Fitness(W,L)*) is the fraction of L's events that W can parse exactly, with a special punishment factor for benefitting the schemas that yield fewer replay errors in fewer traces.

Moreover, the behavioral precision of schema W w.r.t. log L, denoted by $BehPrec(W, L)$, was simply measured as the average fraction of activities that are not enabled by a replay of L in W.

As to schema's complexity, we considered: the numbers of nodes (*#nodes*) and of edges (*#edges*), and the average number of edges per node (*#edgePerNode*).

Test bed As no current approach mixes automated event abstraction and trace clustering, we tested these two facets incrementally, for the sake of comparison.

First, we assessed the ability of our event abstraction method to help discover higher-quality workflow schemas. To this end, we evaluated the workflow schemas extracted by algorithm FHM to an abstract version of the log, obtained by replacing the original events with the event clusters found by `HLPM-mine`, ran without any trace clustering (i.e. by setting $m = 1$). As a term of comparison, we considered three "manual" event abstraction criteria: *(i)* replacing each event with the associated value of `status`, or *(ii)* of `substatus`, and *(iii)* using each distinct event 8-ple as an activity type (`all`).

As to trace clustering, we considered four competitors: algorithm `Actitrac` [8]; the sequence-based and alphabet-based versions of the approach in [4] (`TRMR` and `A-TRMR`, resp.); and the approach in [5] (`KGRAM`), exploiting a k-gram representation. Since all of these competitors lack any mechanisms for abstracting log

cluster	rule
e_1	substatus=assigned ∧ org_line=G199 3rd ∧ org_country=us ∧ resource_country=usa
e_3	substatus=in_progress ∧ org_line=G199 3rd ∧ org_country=us ∧ resource_country=usa
e_4	substatus=in_progress ∧ org_line=G199 3rd ∧ org_country=us ∧ resource_country=poland
e_5	substatus=closed ∧ org_line=G199 3rd ∧ resource_country=poland
e_7	substatus=awaiting_assignment ∧ org_line=G199 3rd ∧ org_country=us

cluster	rule
t_1	$\neg\, e_4 \wedge \neg\, e_9 \wedge \neg\, e_{13}$
t_2	$\neg\, e_4 \wedge e_9$
t_3	$\neg\, e_4 \wedge \neg\, e_9 \wedge e_{13}$
t_4	$e_4 \wedge e_2 \wedge$ month ≤ 5
t_5	$e_4 \wedge e_2 \wedge$ month $> 5 \wedge$ workload ≤ 355
t_6	$e_4 \wedge e_2 \wedge$ month $> 5 \wedge$ workload > 355
t_7	$e_4 \wedge \neg\, e_2 \wedge$ product $\in \{PROD98, PROD96\}$
t_8	$e_4 \wedge \neg\, e_2 \wedge$ product $\in \{PROD428, PROD97\}$

Fig. 3. Some event clusters (left) and trace clusters (right) found by HLPM-mine.

events and for selecting the number of clusters, we provided each of them with the abstraction function and the number of trace clusters found by our approach.

Quantitative Results Table 1 reports the quality scores obtained by our event clustering approach (without trace clustering), here viewed as an enhanced data-driven event-abstraction criterion, compared with the "manual" basic ones described before. To this end, we first partitioned all log events via the Steps 1-3 of algorithm HLPM-mine, and then replaced each event with the label of its respective cluster. All measures were computed by averaging the results of 10 trials, performed each in 10-fold cross-validation.

Interestingly, our event abstraction method gets the best fitness outcome (0.815), and a satisfactory precision score, at the cost of little increase in structural complexity (8 more nodes and 14 more edges) w.r.t. the substatus abstraction, which is the most effective among all 1-attribute abstractions. By the way, only the dummy abstraction all gets higher precision, but it returns overly complex and overfitting workflows.

The benefit of clustering log traces is made clear by Table 2, which reports the quality results obtained by our approach and by the competitors. More precisely, in each clustering test, we computed an overall *Fitness* (resp., *BehPrec*) measure for each method, as the weighted average (with cluster sizes as weights) of the *Fitness* (resp., *BehPrec*) scores received by the workflow schemas induced (with FHM) from all the trace clusters discovered in the test. As all cross-validation trials HLPM-mine (ran with $m{=}\infty$) yielded 8 trace clusters, the same number of clusters was given as input to the competitors.

Notably, HLPM-mine managed to neatly improve the average fitness (0.851) w.r.t. the case where no trace clustering was performed (0.815), and surprisingly outperformed all competitors on this fundamental metrics, despite it can only split the log by way of boolean formulas over trace properties. HLPM-mine also managed to achieve good precision (0.742), compared to its competitors, and to find quite readable workflow schemas —exhibiting, indeed, the lowest average numbers of nodes (10), edges (14) and edges per node (2.6).

Qualitative Results In order to give a more concrete idea of the models that our approach can extract, we ran algorithm HLPM-mine on the entire log (without cross validation), while still setting $m{=}\infty$. As a result, 8 trace clusters were found, and equipped with separate workflow schemas, each describing a distinct

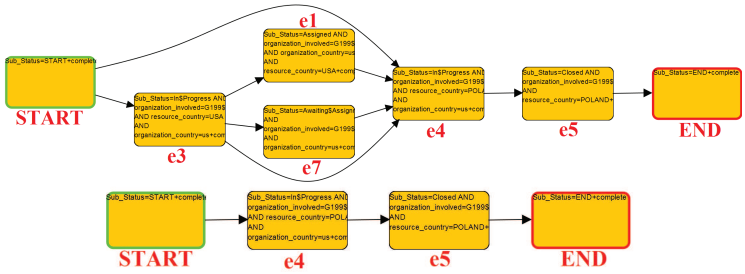

Fig. 4. Workflow schemas induced from trace clusters t_7 (top), and t_8 (bottom).

problem-management scenario. For instance, Figure 4 shows the models of two trace clusters —notice that each node is labelled with the identifier of an event cluster, whose clustering rule can be found in Figure 3 (left).

Besides being more compact than the models discovered without trace clustering (see the last row in Table 1), these schemas help us reckon that problem cases follow different execution scenarios. Specifically, the schema of cluster t_8 captures a particular scenario, where two activities (i.e. event clusters e_4 and e_5) are executed in sequence. Moreover, the trace clustering rules on the right of Figure 3 let us reckon that these scenarios differ both for the value of context factors, and for the occurrence of certain activity types (i.e. event clusters).

6 Discussion and future work

We have presented a clustering-based process discovery method for low-level multi-dimensional logs, where event and trace clusters are used to capture different activity types and process variants, respectively. Tests on a real-life log showed that the approach is able to find high-quality readable workflow models.

Novelty points Several features distinguish our proposal from current process mining solutions, in addition to the very idea of combining activity abstraction and trace clustering — which has only be explored in [4] so far. Fist of all, each event/trace clustering function discovered by our approach is natively encoded in terms of logical rules, which the analyst can easily interpret, to distillate a semantical view of process behavior and of its dependence on relevant properties of process events/cases (and on other context-related factors). Moreover, we removed the common assumption that each log event explicitly refers (or can be easily mapped) to some predefined process activity. In fact, even most current activity abstraction methods [10, 6] rely on the presence of activity labels (possibly defined at a high level of granularity) associated with log events, when trying to aggregate them into higher-level activities (or sub-processes). We pinpoint that no high-level activity types are assumed to be known in advance, differently from [2], where a method was proposed to map log events to a-priori given activity types. The problem faced here is, in fact, more challenging, in that event types are to be learnt inductively from scratch.

Owing to the predictive nature of our models, they can help implement advanced run-time services, beside serving descriptive analyses. For example, one can dynamically assign any novel process case, say c, to one of the discovered process variants (i.e., trace clusters). The workflow schema associated with that cluster can be then presented as a customized (context-adaptive) process map, showing how c could proceed. Moreover, by continuously evaluating the degree of compliance between c and its reference schema deviating can be detected.

Future work Implementing the advanced run-time services above (useful in flexible BPM settings) is left for future work. Inspired by the proposal in [11], we will also try to extend our approach to deal with interleaved logs, which mix traces of different processes without keeping information on which process generated each of them. This would let us release another common assumption —orthogonal to that considered in our work, on the existence of a-priori knowledge on the mapping of log events to process activities— of most process mining approaches. Finally, we plan to test our approach on a wider range of logs.

References

1. van der Aalst, W.M.P., van Dongen, B.F., et al.: Workflow mining: a survey of issues and approaches. Data Knowl. Eng. 47(2), 237–267 (2003)
2. Baier, T., Mendling, J., Weske, M.: Bridging abstraction layers in process mining. Inf. Syst. 46, 123–139 (2014)
3. Blockeel, H., De Raedt, L.: Top-down induction of first-order logical decision trees. Artif. Intell. 101(1-2), 285–297 (1998)
4. Bose, R.P.J.C., van der Aalst, W.M.P.: Trace clustering based on conserved patterns: Towards achieving better process models. In: BPM Workshops. pp. 170–181 (2009)
5. Bose, R.P.J.C., van der Aalst, W.M.P.: Context aware trace clustering: Towards improving process mining results. In: SDM'09. pp. 401–412 (2009)
6. Bose, R.P.J.C., van der Aalst, W.M.P.: Abstractions in process mining: A taxonomy of patterns. In: BPM'09. pp. 159–175 (2009)
7. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: On the role of fitness, precision, generalization and simplicity in process discovery. In: OTM'12, pp. 305–322 (2012)
8. De Weerdt, J., van den Broucke, S., et al.: Active trace clustering for improved process discovery. IEEE Trans. on Knowl. and Data Eng. 25(12), 2708–2720 (2013)
9. Greco, G., Guzzo, A., et al.: Discovering expressive process models by clustering log traces. IEEE Trans. Knowl. and Data Eng. 18(8) (2006)
10. Günther, C.W., Rozinat, A., van der Aalst, W.M.P.: Activity mining by global trace segmentation. In: Proc. of BPM Workshops. pp. 128–139 (2009)
11. Liu, X.: Unraveling and learning workflow models from interleaved event logs. In: ICWS'14. pp. 193–200 (2014)
12. de Medeiros, A.A.: Genetic Process Mining. Phd thesis, TU/e (2006)
13. Weidlich, M., Polyvyanyy, A., et al.: Process compliance analysis based on behavioural profiles. Inf. Syst. 36(7), 1009–1025 (2011)
14. Weijters, A.J.M.M., Ribeiro, J.T.S.: Flexible heuristics miner (FHM). In: CIDM'11. pp. 310–317 (2011)