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

Multi-scenario Analysis and Prediction of Business Processes

Francesco Folino[1], Gianluigi Greco[2],
Antonella Guzzo[3], Luigi Pontieri[1]

RT-ICAR-CS-09-07 **Luglio 2009**

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*

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

Multi-scenario Analysis and Prediction of Business Processes

Francesco Folino[1], Gianluigi Greco [2],
Antonella Guzzo [3], Luigi Pontieri[1]

Rapporto Tecnico N.: **RT-ICAR-CS-09-07**

Data: **Luglio 2009**

---

[1] Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sede di Cosenza, Via P. Bucci 41C, 87036 Rende(CS)
[2] Università degli Studi della Calabria, Dipartimento di Matematica, Via P. Bucci 30B, Rende (CS)
[3] Università degli Studi della Calabria, Dipartimento di Elettronica, Informatica e Sistemistica, Via P. Bucci 41C, Rende (CS)

Multi-scenario Analysis and Prediction of Business Processes

F. Folino[1], G. Greco[2], A. Guzzo[3], and L. Pontieri[1]

ICAR-CNR[1], Via P. Bucci 41C, 87036 Rende, Italy
Dept. of Mathematics[2], UNICAL, Via P. Bucci 30B, 87036, Rende, Italy
DEIS[3], UNICAL, Via P. Bucci 41C, 87036, Rende, Italy
{ffolino,pontieri}@icar.cnr.it, {ggreco}@mat.unical.it, {guzzo}@deis.unical.it

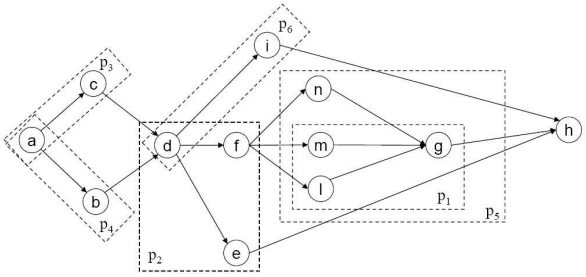
Abstract. Process Mining techniques exploit the information stored in the logs of a variety of transactional systems in order to extract some high-level process model, which can be eventually used for both analysis and design tasks. To deal with the inherent flexibility of real-life processes, recent process mining research evidenced the opportunity of automatically recognizing the different variants, by means of approaches that cluster the input traces based on their behavioral/structural similarity. However, many technical as well as conceptual questions involved in the problem of clustering process traces were not investigated enough, despite their relevance for practical applications. In this paper, we shall focus on two questions arising there: (i) Outlier Detection and (ii) Modelling the association of discovered structural clusters with with other process features (ranging from the executors of the tasks, to the performance metrics, and to the data managed and queried by the transactional system). The former issue clearly impacts on the quality of the clustering results, and requires defining novel outlier detection approaches that are capable to effectively deal with peculiarities of workflow processes, including concurrency and synchronization constructs. As to the latter issue, it would be beneficial to adopt predictive models that allow to exploit additional non-structural information at run-time to foresee as accurately as possible the behavioral class of the current enactment. This advanced capability could be particularly precious in complex and dynamical process management environments. Therefore, in this paper we first address the problem of identifying anomalous traces in the context of process mining applications, and propose a cluster-based outlier detection approach, where a structure-oriented clustering is computed for the logs, while reckoning as outliers those individuals that hardly belong to any of the computed clusters or that belong to clusters whose size is definitively smaller than the average cluster size. We then address the problem of identifying the links between the various structural classes (i.e., the execution scenarios) discovered via the above clustering algorithm and the non-structural features of the process at hand, by introducing a framework for the discovery of a special kind of decision trees, such that the sooner an attribute tends to be known along the course of process enactments, the closer it should appear to the root. Both techniques are synergically applied on two real applicative scenarios in order to prove their efficacy and potentiality.

Keywords: Business Process Intelligence, Process Mining, Clustering, Decision Trees

1 Introduction

In the context of enterprise automation, *process mining* has recently emerged as a powerful approach to support the analysis and the design of complex business processes [24].

In a typical process mining scenario, a set of *traces* registering the sequence of tasks performed along several enactments of a transactional system—such as a Workflow Management (WFM), an Enterprise Resource Planning (ERP), a Customer Relationship Management (CRM), a Business to Business (B2B), or a Supply Chain Management (SCM) system—is given to hand, and the goal is to (semi)automatically derive a model explaining all the episodes recorded in them. Eventually, the "mined" model can be used to design a detailed process schema capable to support forthcoming enactments, or to shed lights into its actual behavior. Thus, process mining is of particular interest when no formal description of the process is available beforehand, or when its observed enactment deviates from the expected one.



s_1 :	abdfnmlgeh	(2)	s_9 :	acdfmlgeh	(1)
s_2 :	abdfelmngh	(2)	s_{10} :	acdih	(10)
s_3 :	abdeflmngh	(3)	s_{11} :	abdih	(8)
s_4 :	abdfmnlgeh	(1)	s_{12} :	afih	(1)
s_5 :	abdeflnmgh	(2)	s_{13} :	ah	(1)
s_6 :	acdfmlgeh	(2)	s_{14} :	aeg	(2)
s_7 :	acdfelmgh	(2)	s_{15} :	adfemh	(1)
s_8 :	acdeflmgh	(3)	s_{16} :	acdfmenlgh	(2)

Fig. 1: A schema W_{ex} (left) and a log L_{ex}(right) – frequencies of (aggregated) traces are shown in brackets.

Traditional process mining approaches focus on capturing the "structure" of the process by way of some sort of control-flow model, which mainly expresses inter-task dependencies via precedence/causality links and other routing constructs, specifying, e.g., the activation/synchronization of concurrent branches, exclusive choices, and loops over all the registered traces. For instance, given the event log (over tasks `a`, `b`, ...`h`) shown in the right side of Figure 1 and consisting of the traces $s_1, ..., s_{14}$, a traditional process mining algorithm would derive an explicative model such as the one depicted in the left side of the same figure, which represents a simplified process schema according to the intuitive notation where precedence relationships are depicted as directed arrows between tasks (e.g., `m` must be executed after `f`, while it can be executed concurrently with `l`). While this kind of approach naturally fits those cases where processes are very-well structured, it would hardly be effective in real-life processes that tend to be less structured and more flexible. Indeed, in such cases, equipping all the traces with one single model would lead to mix different usage scenarios, thereby resulting in a spaghetti-like model, which is rather useless in practice.

To deal with the inherent flexibility of real-life processes, recent process mining research [11, 10, 2, 22, 3, 27] evidenced the opportunity of automatically recognizing the different variants, by means of approaches that cluster the input traces based on their be-

havioral/structural similarity. In particular, much efforts have been spent to define suitable metrics that can be used to compare the various traces and to establish their similarity, which is a pre-requisite for clustering algorithms. For instance, [11, 22] proposed to transform each trace into a vector of a space whose dimensions are one-to-one associated with the frequent substrings emerging from the input logs, whereas [3] considered an edit-distance measure, based on the idea of computing the best possible alignment between the two traces at hand. As a matter of fact, however, beside these issues related to the definition of clustering metrics, many technical as well as conceptual questions involved in the problem of clustering process traces were not investigated in earlier literature, despite their relevance for practical applications. In this paper, we shall focus on two questions arising there:

(1) Outlier Detection. In the case where no exceptional circumstances occur in enactments, clustering approaches for process mining have been proven to be effective in discovering accurate set of process models. However, logs often reflect temporary malfunctions and anomalies in evolutions, whose understanding may help in recognizing critical points in the process that could yield invalid or inappropriate behavior. Indeed, if such exceptional individuals (usually referred to as *outliers* in the literature) were not properly identified, then clustering algorithms will likely mix the actual variants with specific behaviors that are not representative of some usage scenario, but which rather reflect some malfunctioning occurred in the system. Thus, towards improving the quality of the clustering results, it is of utmost importance to define suitable approaches for identifying outliers from a given event log, which are instead currently missing in the literature.

(2) Predictive Models. A tacit assumption in all the approaches to clustering log traces is that the "structure" of each trace reflects some specific behavior of the enactment, so that each cluster can ultimately be associated with a scenario that is characterized by some homogeneous features (ranging from the executors of the tasks, to the performance metrics, and to the data managed and queried by the transactional system). Thus, if such additional non-structural information were available at run-time, the natural question comes into play about whether it might be used to predict the cluster where the current enactment will likely belong to. In other words, one may ask for evidencing the hidden associations between the cluster structure and the underlying non-structural data. Knowing these associations paves, in fact, the way to build forecasting tools (in the spirit, e.g., of [23, 12, 19]) that predict as accurately as possible the behavioral class of the current enactment. This advanced capability could be particularly precious in complex and dynamical process management environments, but mining techniques tailored for its automatic support were not investigated in the literature.

Despite their relevance for practical applications, the problems of singling out outliers from the input traces and of finding predictive models for the clustering results received little attentions so far. The aim of this paper is precisely to complement current research on clustering approaches for process mining applications, and to discuss techniques devoted to provide support in these two contexts.

In more detail, in the first part of the paper, we consider the problem of identifying anomalous traces in the context of process mining applications. To face this problem, we propose to firstly characterize the "normality" of a given set of traces, by mining the (concurrency-aware) structural patterns that frequently occur in the log. Then, we use an outlier detection approach which is *cluster-based*, i.e., it computes a clustering for the logs (where the similarity measure roughly accounts for how many patterns jointly characterize the execution of the traces) and finds outliers as those individuals that hardly belong to any of the computed clusters or that belong to clusters whose size is definitively smaller than the average cluster size.

In the second part of the paper, we address the problem of identifying the links between the various structural classes (i.e., the execution scenarios) discovered via the above clustering algorithm and the non-structural features of the process at hand, in order to build a predictive model on the structure of forthcoming process instances. Technically, the model is conceived as a decision tree. In order to classify as soon as possible novel enactments at run-time, a desirable feature for such a decision tree is that the sooner an attribute tends to be known along the course of process enactments, the closer it should appear to the root. To guarantee such feature, an ad-hoc induction algorithm is also defined and illustrated.

Eventually, in the last part of the paper, the two above techniques are synergically applied on two real applicative scenarios. In particular, experiments are conducted to assess the efficacy of the outlier detection and of the tree induction techniques, by showing how the quality of the clustering and of the predictive models is strongly related to the ability of properly singling out the abnormal circumstances registered in the log.

2 Outlier Detection in Process Mining Applications

2.1 Overview of the Approach

Outlier detection has already found important applications in bioinformatics [1], fraud detection [7], and intrusion detection [4, 13], just to cite a few. The basic observation underlying the various approaches is that abnormality of outliers can not, in general, be defined in "absolute" terms, since outliers show up as kinds of individuals whose behavior or characteristics "significantly" deviate from the normal one(s) that can be inferred through some statistical computation on the data to hand. When extending this line of reasoning towards process mining applications, some novel challenges however come into play:

(C1) On the one hand, looking at the statistical properties of the sequencing of the events might be misleading in some cases. Indeed, real processes usually allow for a high degree of concurrency in the execution of tasks and, hence, a lot of process traces are likely to occur that only differ among each other in the ordering between parallel tasks. As a consequence, the mere application of existing outlier detection approaches for sequential data to process logs may well suffer from a rather high rate of *false positives*, as a notable fraction of task sequences might have very low frequency in the log. As an example, in Figure 6, each of the traces in $\{s_1, ..., s_5\}$ rarely occurs in the log. Yet, it has not to be classified as anomalous, since we may easily see that each of such traces corresponds to a different interleaving for the same enactment, which occurs in 10 out of 43 traces.

(C2) On the other hand, considering the compliance with an ideal schema might lead to *false negatives*, since some trace might well be supported by a model, even though it identifies a behavior that is deviant with respect the one observed in the majority of the traces. As an example, in Figure 6, the trace s_{16} correspond to cases where all the tasks but `b` are executed. Even though this is possible according to the process model on the left, it is yet anomalous since it is registered only in 2 out of 43 traces.

In addition, facing **(C1)** and **(C2)** above is complicated by the fact that the process model underlying a given set of traces is generally unknown and has to be inferred from the data itself. E.g., in our running example, a preliminary question is how we can recognize the abnormality of a trace, without any a-priori knowledge about the model for the given process. Addressing this question and subsequently **(C1)** and **(C2)** is precisely the aim of this section, where an outlier detection technique tailored for process mining applications is discussed. In a nutshell, rather than extracting a model that accurately describes all possible execution paths for the process (but, the anomalies as well), the idea is of capturing the "normal" behavior of the process by simpler (partial) models consisting of *frequent structural patterns*. Then, outliers are identified in a two-steps approach:

- First, we mine the *patterns* of executions that are likely to characterize the behavior of a given log. In fact, our contribution is to specialize earlier frequent pattern mining approaches to the context of process logs, by *(i)* defining a notion of pattern which can effectively characterize concurrent processes by accounting for typical routing constructs arising in process models, and by *(ii)* presenting an algorithm for their identification.
- Second, we use an outlier detection approach which is *cluster-based*, i.e., it computes a clustering for the logs (where the similarity measure roughly accounts for how many patterns jointly characterize the execution of the traces) and finds outliers as those individuals that hardly belong to any of the computed clusters or that belong to clusters whose size is definitively smaller than the average cluster size.

By this way, we will discover, e.g., that traces $s_{11},...,s_{14}$ are not characterized by any of the frequent behaviors registered in the log. Moreover, we will reduce the risk of both false positives (traces are compared according to their characterization in terms of patterns rather than in terms of tasks' sequencing) and false negatives (traces compliant with the model might be seen as outliers, if their behavior is witnessed just in a small group of other traces)—cf. **(C1)** and **(C2)**.

The above techniques are illustrated in Section 2.2, while some basic algorithmic issues are discussed in the subsequent Section 2.3.

2.2 Formal Framework for Outlier Detection

Process-oriented commercial systems usually store information about process enactments by tracing some events related to the execution of the various tasks. By abstracting from the specificity of the various systems, as commonly done in the literature, we may view a *log* L over a set of tasks T as a bag of *traces* over T, where each trace t in L has the form $t[1]t[2]...t[n]$, with $t[i] \in T$ for each $1 \leq i \leq n$. Next, these traces are assumed to be given in input and the problem of identifying anomalies among them is investigated.

Behavioral Patterns over Process Logs. The first step for implementing outlier detection is to characterize the "normal" behavior emerging from a given process log. In the literature, this is generally done by assessing the causal relationships that hold between pairs of tasks (e.g., [21, 14]). However, this is not sufficient to our aims, since abnormality of traces may emerge not only w.r.t. the sequencing of the tasks, but also w.r.t. other more complex constructs such as branching and synchronization. Hence, towards a richer view of process behaviors, we next focus on the identification of those features that emerge as complex patterns of executions.

Definition 1 (S-Pattern). A *structural* pattern (short: *S*-pattern) over a given set T of tasks is a graph $p = \langle T_p, E_p \rangle$, with $T_p = \{n, n_1, \ldots n_k\} \subseteq T$ such that either:

(i) $E_p = \{n\} \times (\{n_1, \ldots n_k\})$ – in this case, p is called a *FORK*-pattern–, or
(ii) $E_p = (\{n_1, \ldots n_k\}) \times \{n\}$ – in this case, p is called a *JOIN*-pattern.

Moreover, the *size* of p, denoted by $size(p)$, is the cardinality of E_p. □

Notice that, as a special case, an *S*-pattern with unitary size is both a *FORK*-pattern and a *JOIN*-pattern, and simply models a causal precedence between two given tasks. This is, for instance, the case of patterns p_3, p_4, and p_5 in Figure 6. Instead, higher size patterns account for fork and join constructs, which are typically meant to express parallel execution (cf. p_1) and synchronization (cf. p_2), respectively, within concurrent processes.

The crucial question is now to formalize the way in which patterns emerge for process logs.

Definition 2 (Pattern Support). Let t be a trace and let $p = \langle T_p, E_p \rangle$ be an S-pattern. We say that t *complies with* p, if **(a)** t includes all the tasks in T_p and **(b)** the projection of t over T_p is a topological sorting of p, i.e., there not exist two positions i, j inside t such that $i < j$ and $(t[j], t[i]) \in E_p$. Then, the support of p w.r.t. t, is defined as:

$$supp(p,t) = \begin{cases} \min_{(t[i],t[j]) \in E_p} e^{-|\{t[k] \notin T_p | i<k<j\}|}, & \text{if } t \text{ complies with } p \\ 0, & \text{otherwise.} \end{cases}$$

This measure is naturally extended to any trace bag L and pattern set P as follows: $supp(p,L) = \frac{1}{|L|} \times \sum_{t \in L} supp(p,t)$ and $supp(P,t) = \frac{1}{|P|} \times \sum_{p \in P} supp(p,t)$. □

In words, a pattern p is not supported in a trace t if some relation of precedence encoded in the edges of p is violated by t. Otherwise, the support of p decreases at the growing of the minimum number of spurious tasks (i.e., $\{t[k] \notin T_p \mid i < k < j\}$) that occur between any pair of tasks in the endpoints of the edges in p.

Example 1. Consider again the example shown in Figure 6. It is clear that all traces corresponding to any of the sequences $s_9, ..., s_{15}$ do not comply with p_1. For the remaining traces, the application of the support function defined in [9] gives the following results:

$$supp(p_1, s_1) = supp(p_1, s_6) = supp(p_1, s_7) = supp(p_1, s_8) = supp(p_1, s_9) = e^{-0} = 1$$
$$supp(p_1, s_2) = supp(p_1, s_3) = supp(p_1, s_4) = supp(p_1, s_5) = e^{1} = 0.368$$
$$supp(p_1, s_{16}) = e^{-2} = 0.135$$

Therefore, given the frequencies in Figure 6, the support of p_1 w.r.t. the whole log thus is 0.307. By similar calculations we also have that p_5 gets full support (i.e 1) by $s_1, ..., s_5$, and a support of 0.368 by s_{16}, for a total of 0.249 against the whole log. ◁

While at a first sight this notions may appear similar to classical definitions from frequent pattern mining research, some crucial and substantial differences come instead into play. Indeed, the careful reader may have noticed that our notion of support is not *anti-monotonic* regarding graph containment. This happens because adding an edge of the form (x, y) to a given pattern may well lead to increase its support, since one further task (either x or y) may be no longer viewed as a spurious one. Consequently, the space of all the possible S-patterns does not form a lattice, and classical *level-wise* approaches cannot be used to single out those patterns whose support over a log L is greater than a given threshold σ, hereinafter called σ-*frequent* patterns.

In addition, differently from many pattern mining approaches, the frequency of a pattern p is not necessarily an indication of its relevance in the regard of modeling the process behavior. In particular, when comparing two σ-frequent patterns p_1 and p_2 such that p_1 is a subgraph of p_2, we can safely focus on p_2 if its frequency is not significantly different from the one of p_1; otherwise, i.e., if p_1 happens to be much more frequent than p_2, the subpattern p_1 has also its own interest in the characterization of the process. This is formalized below.

Definition 3 (Interesting Patterns). Let L be a log, and σ, γ be two real numbers. Given two S-patterns p_1 and p_2, we say that p_2 γ-*subsumes* p_1, denoted by $p_1 \sqsubseteq_\gamma p_2$, if p_1 is a subgraph of p_2 and $supp(p_1, L) - supp(p_2, L) < \gamma \times supp(p_2, L)$. Moreover, an S-pattern p is (σ, γ)-*maximal* w.r.t. L if **(a)** p is σ-frequent on L and **(b)** there is no other S-pattern p' s.t. $size(p') = size(p) + 1$, p' is σ-frequent on L, and $p \sqsubseteq_\gamma p'$. □

Example 2. Let us consider the patterns p_5 and p_1 in Figure 6, σ=0.1 and γ=0.2. Then, even though p_1 is contained in p_5 (and both of them are frequent), the former is still maximal as $(supp(p_1, L) - supp(p_5, L))/supp(p_5, L) = (0.307 - 0.249)/0.249 = 0.233 > \gamma$. Therefore, this sub-pattern still encodes interesting knowledge as it captures a far more frequent way of executing the tasks `m` and `g` than the one expressed by its super-pattern p_5. Conversely, no subgraph of p_2 is (σ, γ)-maximal, being the support of all these patterns lower than the support of p_2. ◁

Clusters-based Outliers. Once that "normality" has been modeled by means of the discovery of interesting patterns, we can then look for those individuals whose behavior deviates from the normal one. To this end, the second step of our outlier detection approach is based on a *coclustering* (see, e.g., [5]) method for simultaneously clustering both patterns and traces, on the basis of their mutual correlation, as it is expressed by the measure $supp$.

Intuitively, we look for associating pattern clusters with trace clusters, so that outliers emerge as those individuals that are not associated with any pattern cluster or that belong to clusters whose size is definitively smaller than the average cluster size. Abstracting from the specificity of the mining algorithm (discussed in Section 2.3), the output of this method is formalized below.

Definition 4 (Coclusters and Outliers). An α-*coclustering* for a log L and a set P of S-patterns is a tuple C=$\langle \mathcal{P}, \mathcal{L}, \mathcal{M} \rangle$ where:

- $\mathcal{P}$=$\{p_1, ..., p_k\}$ is a set of non-empty P's subsets (named *pattern clusters*) s.t. $\bigcup_{j=1}^{k} p_j$=P;
- $\mathcal{L}$=$\{l_1, ..., l_h\}$ is a set of non-empty disjoint L's subsets (named *trace clusters*) such that $\bigcup_{i=1}^{h} l_i = \{t \in L \mid \exists p_i \in \mathcal{P} \text{ s.t. } supp(p_i, t) \geq \alpha\}$;
- $\mathcal{M} : \mathcal{P} \mapsto \mathcal{L}$ is an bijective function that associates each pattern cluster p_j to a trace cluster l_i and vice-versa, i.e., $l_i = \mathcal{M}(p_j)$ and $p_j = \mathcal{M}^{-1}(l_i)$.

Given two real numbers α, β in [0..1], a trace $t \in L$ is an (α, β)-*outlier* w.r.t. an α-coclustering $C = \langle \mathcal{P}, \mathcal{L}, \mathcal{M} \rangle$ if either **(a)** $t \notin \bigcup_{i=1}^{h} l_i$, or **(b)** $|l_i| < \beta \times \frac{1}{|\mathcal{L}|} \sum_{l_j \in \mathcal{L}} |l_j|$, where $t \in l_i$. □

In words, we define outliers according to a number of clusters, discovered for both traces and patterns based on their mutual correlations, which represent different behavioral classes. More specifically, two different kinds of outlier emerge; indeed, condition

(a) deems as outlier any trace that is not assigned to any cluster (according to the minimum support α), while condition **(b)** estimates as outliers all the traces falling into small clusters (smaller than a fraction β of the average clusters' size).

Example 3. Let us consider again the example log and patterns shown in Figure 6. By evaluating the support measure in Definition 2, one may notice that the traces corresponding to $s_1, ..., s_5$ highly support patterns p_2, p_4 and p_5, while s_6, s_7 do the same with both patterns p_1 and p_3. Moreover, s_8 highly supports both p_3 and p_6, whereas s_9 is strongly associated with both p_4 and p_6. Finally, sequence s_{14} is associated with all of the patterns in Figure 6 but p_4. By using some suitable co-clustering method on the correlations between these patterns and log traces, one should hence be able to identify five trace clusters: one corresponding to the sequences $s_1, ..., s_5$; one for $s_6, ..., s_9$, one for s_{10}; one further for the trace s_{11}, and the last for s_{16}. All the other traces would be hence perceived as outliers, for they are not correlated enough with any of these frequent behavioral patterns. A special case concerns the last sequence s_{16}. In actual fact, the above sketched clustering approach would originate a separate cluster, which just consists of the two traces that correspond to s_{16}. However, this cluster reflects a somewhat rare behavioral scheme (evidenced by only 2 of 43 traces), and should not be considered when modelling the main behavioral classes of the process. Clearly, this can be accomplished by properly setting the threshold β, controlling the minimal cluster size. $\triangleleft$

2.3 An Algorithm for Detecting Outliers in a Process Log

In this section, we discuss an algorithm, named `structuralClustering`, for singling out a set of outliers, based on the computation scheme and the framework described so far. The algorithm is shown in Figure 2: Given a log L, a natural number *pattSize* and four real thresholds σ,γ, α and β, it first employs the function `FindPatterns` to compute a set P of (σ,γ)-maximal S-patterns, while restricting the search to patterns with no more than *pattSize* arcs. Then, an α-coclustering for L and P is extracted with the function `FindCoClusters` (Step 2). The following steps are just meant to build a set U of traces that are (α,β)-outliers w.r.t. this coclustering, by checking the conditions in Definition 4 on every trace. Eventually, the (α,β)-outliers are returned together with the set of trace clusters (from which such outliers are removed). Clearly enough, the main computation efforts hinge on the functions `FindPatterns` and `FindCoClusters`, which are thus thoroughly discussed next, in two separate subsections.

Function `FindPatterns` The main task in the discovery of (σ,γ)-maximal S-patterns is the mining of σ-frequent S-patterns, as the former S-patterns directly derive from the latter ones. Unfortunately, a straightforward level-wise approach cannot be used to this end, since the support *supp* is not anti-monotonic w.r.t. pattern containment. To face this problem, `FindPatterns` firstly exploits a relaxed notion of support (denoted $supp'$)

Input: A log L, an upper bound $pattSize \in \mathbb{N}^+$ for pattern size, and four real numbers σ, γ, α and β
Output: A set of (α,β)-outlier, and set of trace clusters;
Method: Perform the following steps:

1 $P :=$ `FindPatterns` $(L, pattSize, \sigma)$;
2 $\langle \mathcal{P}, \mathcal{L} = \{l_1, ..., l_h\}, \mathcal{M} \rangle :=$ `FindCoClusters` (L, P, α);
3 $U := \emptyset$; $\quad avgSize := \frac{1}{|\mathcal{L}|} \sum_{l_j \in \mathcal{L}} |l_j|$;
4 **for each** trace t in L **do**
5 **if** $t \notin \bigcup_{i=1}^{h} l_i$, **or** $|l_i| < \beta \times \frac{1}{h} \sum_{l_j \in \mathcal{L}} |l_j|$, where $t \in l_i$ **then** $U := U \cup \{t\}$;
6 **return** U, and $\mathcal{L}_* = \{l_i \mid l_i \in \mathcal{L} \wedge |l_i| \geq \beta \times \frac{1}{h} \sum_{l_j \in \mathcal{L}} |l_j|\}$;

Function `FindPatterns` (L: log; $pattSize$: natural number; σ: real number): set of S-patterns;

P1 Compute the set $L_1 = \{p$ is an S-pattern $\mid supp'(p,L) \geq \sigma$ and $size(p) = 1\}$ in a scan of L;
P2 $k := 2$; $R := \emptyset$
P3 **repeat**
P4 $Cand_k := generateCandidates(L_{k-1}, L_1)$;
P5 Compute $supp(p,L)$ and $supp'(p,L)$ for each $p \in Cand_k$ through a scan of L;
P6 $L_k := \{p \in Cand_k \mid supp'(p,L) \geq \sigma\}$; *// filter out "unfrequent" patterns*
P7 $R := R \cup \{p \in L_{k-1} \mid \nexists p' \in L_k$ s.t. $p \sqsubseteq_\gamma p'\}$; *// select (σ,γ)-maximal patterns (cf. Def. 3)*
P8 $k := k+1$;
P9 **until** $L_k = \emptyset$ **or** $k+1 = pattSize$;
P10 **return** R;

Function `FindCoClusters` (L: log; P: S-patterns; α: real number): α-coclustering;

C1 **for each** pair of patterns p_i, p_j in P **do** $M(i,j) := \frac{|\{t' \mid supp(p_i,t') \geq \alpha \wedge supp(p_j,t') \geq \alpha\}|}{|\{t' \mid supp(p_i,t') \geq \alpha \vee supp(p_j,t') \geq \alpha\}|}$
C2 Compute a partition P_{mcl} of P by applying the *MCL* clustering algorithm to M;
C3 $\mathcal{L} := \emptyset$; $\mathcal{P} := \emptyset$; $\mathcal{M} := \emptyset$;
C4 **for each** trace t in L
C5 $p^t := \bigcup_{p \in P_{mcl}} \{p \mid supp(p,t) \geq \alpha\}$;
C6 **if** P contains p^t *// cluster p^t already exists and is hence associated with some trace cluster*
C7 Let $l^t = \mathcal{M}(p^t)$ be the cluster currently associated with p^t, and $l^t_{new} = l^t \cup \{t\}$;
C8 $\mathcal{L} := \mathcal{L} - \{l^t\} \cup \{l^t_{new}\}$; $\mathcal{M}(p^t) := l^t_{new}$;
C9 **else**
C10 $\mathcal{L} := \mathcal{L} \cup \{\{t\}\}$; $\mathcal{P} := \mathcal{P} \cup \{p_t\}$; $\mathcal{M}(p^t) := \{t\}$;
C11 **end if**
C12 **end for**
C13 **return** $\langle \mathcal{P}, \mathcal{L}, \mathcal{M} \rangle$;

Fig. 2: **Algorithm `structuralClustering`**

which optimistically decreases the counting of spurious tasks by a "bonus" that depends on the size of the pattern at hand: the lower the size the more the bonus. More precisely, within Definition 2, for each arc $(t[i], t[j])$ in p, we replace the term $|\{t[k] \notin T_p \mid i < k < j\}|$ with $min\{\ |\{t[k] \notin T_p \mid i < k < j\}|, pattSize - size(p)\}$. The reason for this is that, in the best case, each of the $pattSize - size(p)$ arcs that might be added to p, along the level-wise computation of patterns, will just fall between i and j.

It can be shown that function $supp'$ is both anti-monotonic and "safe", in that it does not underestimate the actual support of candidate patterns. Therefore, based on it we have implemented a level-wise approach: After building (in Step P1) the basic set L_1 of frequent S-patterns with size 1 (i.e., frequent task pairs), an iterative scheme is used to incrementally compute any other set L_k, for increasing values of the pattern size k (Steps P4–P8), until either no more patterns can be generated or k reaches the upper

bound given as input. In more detail, for each $k > 1$, we first generate the set $Cand_k$ of k-sized candidate patterns, by suitably extending the patterns in L_{k-1} with the ones in L_1, by means of function *generateCandidates* (Step P4). The set L_k is then filled only with the candidate patterns in $Cand_k$ that really achieve an adequate support in the log (Steps P5- P6). By construction of $supp'$, we are then guaranteed that L_k includes (at least) all σ-frequent S-patterns with size k.

Eventually, by a straightforward application of Definition 3 to the patterns in L_{k-1} and L_k, we can single out all (σ,γ)-maximal S-patterns with size $k-1$, and add them to the set R, the ultimate outcome of `FindPatterns`. In fact, in Step P7 the exact function $supp$ is actually used for checking (σ,γ)-maximality.

Function `FindCoClusters` The function `FindCoClusters` illustrates a method for coclustering a log and its associated set of S-patterns. Provided with a log L, a set P of S-patterns and a threshold α, the function computes, in a two-step fashion, an α-coclustering $\langle \mathcal{P}, \mathcal{L}, \mathcal{M} \rangle$ for L and P, where $\mathcal{P}$ (resp., $\mathcal{L}$) is the set of pattern (resp., trace) clusters, while $\mathcal{M}$ is a mapping from $\mathcal{P}$ to $\mathcal{L}$.

At start, a preliminary partition P_{mcl} of P is built by applying a clustering procedure to a similarity matrix S for P, where the similarity between two patterns p_i and p_j in P provides a sort of estimation for the likelihood that p_i and p_j occur in the same log trace. More specifically, these similarity values are computed (Step C1) by regarding $supp$ as a contingency table over P and L (i.e., (p,t) measures the correlation between the pattern p and the trace t), and by filtering out low correlation values according to the threshold α. Clearly, different classical clustering algorithms could be used to extract P_{mcl} out of the matrix M (Step C2). In fact, we used an enhanced implementation of the *Markov Cluster Algorithm* that achieves good results on several large datasets [6], and selects the number of clusters autonomously.

In the second phase (Steps C3-C13), the preliminary clustering P_{mcl} of the patterns is refined, and yet used as a basis for simultaneously clustering the traces of L: new, "high order" pattern clusters are built by merging together basic pattern clusters that relate to the same traces. More precisely, each trace t in the log induces a pattern cluster p^t, which is the union of all the (basic) clusters in P_{mcl} that are correlated enough to t, still based on the function $supp$ and the threshold α. It may happen that the cluster p^t is already in $\mathcal{P}$, for it was induced by some other traces; in this case we retrieve, by using the mapping $\mathcal{M}$, the cluster l^t containing these traces (Step C7), and extend it with the insertion of t (Step C8). Otherwise, we save a new trace cluster, just consisting of t, in $\mathcal{L}$, and update $\mathcal{M}$ to store the association between this new cluster and p^t, which is stored as well in $\mathcal{P}$ as a novel pattern cluster (Step C10).

Before leaving this section, it is worth observing that the `structuralClustering` algorithm can be implemented without importing the input log as a whole into the main memory. Indeed, the input log can just be scanned k times for finding patterns of size k,

plus two further times for building matrix M and for assigning each trace to the various clusters (Steps C4-C12). Thus, main memory computation is just limited to the clustering of the frequent patterns (whose number is generally small compared with the input log—in any case, one usually desire to focus on the most frequent ones). This property guarantees potential scaling over huge datasets.

3 Discovery of Context-based Predictive Models

After that a set $\mathcal{L}_*$ of trace clusters has been computed, e.g., by means of the approach discussed in the previous section, the natural question comes into play about whether we can find a model predicting the membership into the various clusters based on the (non-structural) data available for the process instances at hand. In this section, we shall explore this issue, by conceiving the predictive model as a decision tree. In particular, in Section 3.1, we shall formally describe the kind of context information that is assumed to be available in the input log, and the features that should be enjoyed by the decision trees we would like to associate with the trace clusters . A computation method allowing to extract such a models is then illustrated in detail in Section 3.2.

3.1 Formal Framework for the Induction of Predictive Models

In principle, process logs may contain a wide range of information about process executions. The notion of log traces considered so far is therefore extended next in order to represent context data associated with the execution of tasks. To this end, we assume the existence of a set of process attributes $A = \{a_1, \ldots, a_n\}$, and we assume that each attribute is associated with one single task, referred to as $task(a_i)$ in the following. In particular, case attributes can be associated with the starting (or final) task of the process. Moreover, for ease of notation, for any attribute a and its corresponding task t (i.e. $t = task(a)$), we will sometime refer to a as $t.a$, in order to represent its association with t in a compact and intuitive enough manner. Each attribute $a \in A$ is also equipped with a domain of values, denoted by $dom(a)$.

At run-time, the enactment of the process will cause the execution of a sequence of tasks, where for each task t being executed, the set of all its activities will be mapped to some values taken from the respective domains. This is formalized below.

Definition 5 (Data-Aware Logs). Let T be a set of tasks and let A be a set of process attributes. A *data-aware* log over T and A is a tuple $\langle L, data \rangle$ where L is a log over T, and where $data$ is a function mapping each trace $t \in L$ into a set of pairs $data(t) = \{(a_1, v_1), \ldots, (a_q, v_q)\}$ such that $v_i \in dom(a_i)$ for each $i \in \{1, ..., q\}$, and$\{a_1, ..., a_q\} = \{a \in A \mid task(a) = t[j], \text{for some task } t[j] \in T\}$. □

In the following, we assume that the set $\mathcal{L}_*$ of trace clusters at hand has been built from a data-aware process log L. Thus, based on the knowledge of the data associated

with the execution of the various traces, it is our aim to build a decision tree that can be used to predict membership into the clusters for forthcoming enactments.

Definition 6 (Data-Aware Decision Tree). Let $\mathcal{L}_*$ be a set of trace clusters (for a data-aware process log) over a set T of tasks and a set A of associated attributes. Then, a *data-aware decision tree* (shortly, *DADT*) for $\mathcal{L}_*$ is a triple $\mathcal{D} = \langle H, \texttt{attr}, \texttt{split} \rangle$ such that:

- $H = (N, E)$ is a rooted tree, where N and E denote the set of nodes and the set of (parent-to-child) edges, respectively;
- $\texttt{attr}$ is a function mapping each non-leaf node v in N to an attribute in A;
- $\texttt{split}$ is a function associating each edge from v to w (where w is a child of v) with a propositional formula on $\texttt{attr}(v)$. □

Since we are interested in predicting the happening of behavioral classes based on context data, a desirable property of a *DADT* concerns its ability to take care of the task precedences holding over these classes. To formalize this concept, we need some additional technical definitions first.

We say that a trace t is *active* in a node $v \in N$ of a *DADT* $\mathcal{D} = \langle H, \texttt{attr}, \texttt{split} \rangle$, if t satisfies all the $\texttt{split}$ tests defined in the path from the root of H to v. For a threshold $\sigma \in [0..1]$, we say that a cluster $l \in \mathcal{L}_*$ is σ-*active* in a node $v \in N$ if $|\{t \in l \mid t \text{ is active in } v\}|/|l| > \sigma$. The restriction of $\mathcal{L}_*$ to the clusters that are σ-active in v is denoted by $\mathcal{L}_*(\sigma, v)$. Moreover, for two tasks s and s', we say that s σ-*precedes* s' in l, denoted by $s \prec^l_\sigma s'$, if there is a trace $t \in l$ such that $s = t[i]$ and $s' = t[j]$ with $i < j$, and there is no trace $\bar{t} \in l$ such that $s = \bar{t}[\bar{i}]$ and $s' = \bar{t}[\bar{j}]$ with $\bar{i} > \bar{j}$.

Definition 7 (Temporal Compliance of a *DADT*). Let $\mathcal{D} = \langle H, \texttt{attr}, \texttt{split} \rangle$ be a *DADT* for the data-aware log $\mathcal{L}_*$, and let σ be a threshold in $[0..1]$. We say that $\mathcal{D}$ is σ-*compliant* w.r.t. $\mathcal{L}_*$ if for each pair of nodes v and v' of H such that v' is an ancestor of v, it holds that or each σ-active cluster $l \in \mathcal{L}_*(\sigma, v)$, either:

(a) $task(\texttt{attr}(v)) \prec^l_\sigma task(\texttt{attr}(v'))$ does not hold, or
(b) there is an ancestor v'' of v' such that $task(\texttt{attr}(v'')) = task(\texttt{attr}(v))$. □

In words, condition (a) states that we cannot split a node v of the *DADT* by using an attribute of a task t, if an ancestor of v is associated with an attribute of a task that is usually executed after t (w.r.t. the behavioral clusters in $\mathcal{L}_*(\sigma, v)$. This constraint is however relaxed by the condition (b), which allows to reuse the attributes of a task associated with v'' in whichever node of the tree rooted in v''. These two constraints therefore guarantees that σ-compliant *DADT*s are suitable models to support on-the-fly prediction. The question moves therefore on how we can mine a σ-compliant *DADT*, based on the clustering $\mathcal{L}_*$ at hand. This question is faced in the following section.

3.2 An Algorithm for Inducing a DADT Model

Several decision-tree induction approaches are already available in the literature (see, e.g., [15, 18]) that might be used, in principle, to built a σ-compliant *DADT*. However, by straightforwardly integrating the σ-compliance constraint into such approaches, we risk obtaining a *DADT* tree of poor accuracy. Consider, as an example, the extreme case where an attribute of the final task, say *e*, is chosen for performing the first split of the training set, and consequently associated with the root of the decision tree – assuming that all process instances finished with task *e* and that a top-down, recursive, partition scheme is adopted for inducing the tree. Indeed, in such a case, Definition 7 would allow to further partition the training set based only on attributes of task *e*, since attributes of other tasks (which precede *e* in all log traces) cannot appear in any descendants of the root.

To face the problem above, we modify the greedy split-selection criterion used by classical decision-tree learning algorithms by introducing a bias towards attributes of tasks that were executed in earlier phases of past process enactments. This is mainly accomplished by considering an ad-hoc attribute-scoring function for selecting split tests, which ranks process attributes based on their capability to discriminate the clusters yet supporting on-the-fly prediction.

An algorithm for inducing a σ-compliant *DADT* according to the strategy sketched above is illustrated in Figure 3. The algorithm starts building a preliminary *DADT* that just consists of one node (named *r* in the figure), gathering all log traces (indeed, the set *L* contains the traces of all clusters given in input). Then (line 2) a decision tree is built in a top-down manner, via a recursive partitioning procedure, named `growDT` , which will be discussed in detail later. Once such a (possibly large and overfitted) decision tree has been built, a pruning procedure (like in the J48 implementation of algorithm C4.5 [18]) is exploited to improve its capability to make accurate prediction over new process instances. The pruned DADT model is returned as the ultimate outcome of the algorithm eventually.

Procedure `GrowDT` Let us now provide more details on the recursive procedure `growDT` , which encodes the core induction method for eventually yield a DADT model. The procedure takes as input a data-aware decision tree $\mathcal{D}$, the leaf node *v* and its associated set *S* of traces, which are to be considered for being split, and the original set L_* of (structural) trace clusters. After checking (in step `B1`) whether *v* contains a significant number (according to the cardinality threshold *minCard*) of training instances, the procedure searches for a (locally) optimal way of partitioning these instances (steps `B3`-`B4`). The split test for the node *v* is chosen greedily, by selecting the attribute that receives the highest value by a split quality metrics *score*.

For each attribute *a* such a split score is computed via a linear combination (with a coefficient $\omega \in [0,1]$) of two components:

Input: A set $\mathcal{L}_*$ of trace clusters over tasks T and attributes A, a set $A' \subseteq A$ of attributes, an integer number *minCardinality* ≥ 1 and two real numbers σ and ω;
Output: A σ-compliant *DADT* for $\mathcal{L}_*$;
Method: Perform the following steps:

```
1  let L = ∪_{l_i ∈ L} C_i;
2  create a DADT D s.t. D.H = ⟨{r}, ∅, r⟩ — functions D.attr , D.split  and D.prob  will be defined later
3  growDT (D,r,L,L);
4  pruneDT (D,L);
5  return D;
```

Procedure growDT ($\mathcal{D}$: a *DADT*, v: a $\mathcal{D}$'s node, S: a set of traces; $\mathcal{L}_*$: a set of trace clusters);

```
B1   if |S| ≥ minCard
B2      let L_σ = {l_i ∈ L s.t. |l_i ∩ S| ≥ σ · |S|};
B3      compute score(a) = ω · g(a,S) + (1 − ω) · ep(a,S,L_σ), ∀a ∈ A';
B4      let s*=max_{a∈A}{score(a)}, a*=argmax_{a∈A}{score(a)}, and τ* be the split formula evaluated for a*;
B5      if ω < 1 and checkCompliance  (task(a*), D, v, L_σ)
B6         D.split (v) := τ*;  D.attr (v) := a*;  D.prob (v,l_i) := |S ∩ l_i|/|S|, for each l_i ∈ L_*;
B7         let S_1,...,S_k be the partition of S obtained by applying the test τ* to S;
B8         add k new nodes v_1,...,v_k in D.H as children of v;
B9         for j=1..k  growDT (D,v_j,S_j,L_*);
B10     end if
B11  end if
```

Fig. 3: Algorithm DADT-Induction

- a predictiveness measure, denoted by $g(a,S)$ and computed through classical Gain Ratio measure [16, 17], which mainly founds on evaluating the reduction of information entropy that descends from splitting S according to some suitable split formula over a[1].
- an ad-hoc score, denoted by $ep(a),S,L_\sigma)$, which tries to take account of dynamical aspects of the process, by introducing a bias towards attributes that are associated with tasks that occur earlier in the traces corresponding to the clusters of $\mathcal{L}_*$ that are correlated with v significantly.

More precisely, denoting by $\mathcal{L}_\sigma(v)$ the set of $\mathcal{L}$'s clusters that are significantly represented in S according to minimal frequency threshold σ (cf. Line B2), the latter score is computed as follows:

$$ep(a,S,\mathcal{L}) = \frac{1}{|S|} \sum_{l \in \mathcal{L}} \frac{|l| \cdot |succ(task(a),l)|}{|tasks(l)|}$$

where $tasks(l)$ simply indicates the set of tasks that appear in the traces of cluster l, while $succ(task(a),l)$ denotes the number of tasks in $task(l)$ that follows $task(a)$ according to the ordering relationship $\prec^l_\sigma$, i.e. $succ(task(a),l) = \{t' \in tasks(l) \mid t \prec^l_\sigma t'\}$.

[1] A formula yielding a distinct outcome for each possible value is considered for nominal (discrete) attributes. Conversely, in the case of a numeric (ordinal) attribute, the same heuristic method as J48 is exploited to find a binary partition of its domain of into two continuous ranges

We pinpoint that when making *score* coincide with the Gain Ratio measure (i.e., when $\omega = 1$), it may happen that the check performed by `checkCompliance` arrests the growth of the tree, without allowing the clusters in v to be separated neatly enough. It is just such an undesirable effect that we want to prevent by correcting a classical (purity-based) selection criterion through the *ep* score.

Once a (locally) optimal attribute a^* has been chosen for splitting the traces in S, the `checkCompliance` function is invoked to verify that the constraints in Definition 7 are satisfied (Step B5). Indeed, the application of this function to the parameters a^*, $\mathcal{D}$, v, and $\mathcal{L}_\sigma$ will return `false` iff *(i)* there is an ancestor v' of v in $\mathcal{D}$ such that $task(v')$ precedes a^* in some cluster of C_σ, and *ii* there is no ancestor v'' of v' in $\mathcal{D}$ s.t. $task(v'') = a^*$. Notice that such a test can be speeded up by maintaining some compact representation of relevant task precedences (w.r.t. threshold σ) for each of the behavioral clusters in the set $\mathcal{L}$ given as input to the algorithm. To this purpose, one could well think of resorting to some kind of workflow model (possibly discovered through classical process mining techniques, such as those presented in [24]). Since the compliance test is done only when $\omega < 1$, the behavior of algorithm `DADT-Induction` is made to coincide with that of traditional decision tree learning algorithms in the case $\omega = 1$.

In the case the check performed by `checkCompliance` is passed successfully, the current (leaf) node v is mapped to both the selected split formula τ^* and the associated attribute a^*, by suitably updating the functions *split* and `attr` of the *DADT* $\mathcal{D}$ (line B6); moreover, the joint probability value $\mathcal{D}.\texttt{prob}(v, l_i)$ is estimated, for each cluster l_i, as the percentage of v's instances that belong to l_i.

The decision tree is then expanded by adding as many children of v as the groups $S_1 \ldots S_k$ of traces produced by applying the partition formula τ^* to S (lines B7-B8). Finally, the procedure `growDT` is recursively invoked over each new node v_i, and its corresponding set of traces S_i.

4 Putting It All Together: A Toy Application Example

This section describes a complete application of the approach introduced so far to our running example. In order to show, in particular, the discovery of data-aware predictive models, a refined representation of the log of Figure 6 is given in Figure 4. Based on the notation of Definition 5, each trace in this latter figure, generated from just one enactment case, corresponds both to a sequence of tasks and to a number of non-structural data (encoded in terms of attribute-value pairs). By the way, notice that this log concerns the processing of liability claims in an insurance company, and was basically inspired to the running example used in [19].

The behavior of the underlying process can be summarized as follows: after registering data about the claim (**a**, Register claim), either a full check (**c**, Check all) or a shorter one, only involving policy data (**b**, Check policy only), is performed. Once the claim has

trace ID	task sequence	data
t1	s1:abdfnmlgeh	{(a.Amount,1000),(a.PolicyType,premium),(d.Status,approved)}
t2	s1:abdfnmlgeh	{(a.Amount,1050),(a.PolicyType,premium),(d.Status,approved)}
t3	s2:abdfelmngh	{(a.Amount,5000),(a.PolicyType,premium),(d.Status,approved)}
t4	s2:abdfelmngh	{(a.Amount,500),(a.PolicyType,premium),(d.Status,approved)}
t5	s3:abdeflmngh	{(a.Amount,495),(a.PolicyType,premium),(d.Status,approved)}
t6	s3:abdeflmngh	{(a.Amount,500),(a.PolicyType,normal),(d.Status,approved)}
t7	s3:abdeflmngh	{(a.Amount,480),(a.PolicyType,normal),(d.Status,approved)}
t8	s4:abdfmnlgeh	{(a.Amount,6000),(a.PolicyType,premium),(d.Status,approved)}
t9	s5:abdeflnmgh	{(a.Amount,6200),(a.PolicyType,premium),(d.Status,approved)}
t10	s5:abdeflnmgh	{(a.Amount,5800),(a.PolicyType,premium),(d.Status,approved)}
t11	s6:acdfmlgeh	{(a.Amount,500),(a.PolicyType,normal),(d.Status,rejected)}
t12	s6:acdfmlgeh	{(a.Amount,490),(a.PolicyType,normal),(d.Status,rejected)}
t13	s7:acdfelmgh	{(a.Amount,600),(a.PolicyType,premium),(d.Status,rejected)}
t14	s7:acdfelmgh	{(a.Amount,610),(a.PolicyType,premium),(d.Status,rejected)}
t15	s8:acdeflmgh	{(a.Amount,615),(a.PolicyType,premium),(d.Status,rejected)}
t16	s8:acdeflmgh	{(a.Amount,605),(a.PolicyType,premium),(d.Status,rejected)}
t17	s8:acdeflmgh	{(a.Amount,620),(a.PolicyType,premium),(d.Status,rejected)}
t18	s9:acdfmlgeh	{(a.Amount,400),(a.PolicyType,premium),(d.Status,rejected)}
t19	s10:acdih	{(a.Amount,501),(a.PolicyType,normal),(d.Status,approved)}
t20	s10:acdih	{(a.Amount,555),(a.PolicyType,normal),(d.Status,approved)}
t21	s10:acdih	{(a.Amount,560),(a.PolicyType,normal),(d.Status,approved)}
t22	s10:acdih	{(a.Amount,565),(a.PolicyType,normal),(d.Status,approved)}
t23	s10:acdih	{(a.Amount,570),(a.PolicyType,normal),(d.Status,approved)}
t24	s10:acdih	{(a.Amount,575),(a.PolicyType,normal),(d.Status,approved)}
t25	s10:acdih	{(a.Amount,580),(a.PolicyType,normal),(d.Status,approved)}
t26	s10:acdih	{(a.Amount,585),(a.PolicyType,normal),(d.Status,approved)}
t27	s10:acdih	{(a.Amount,590),(a.PolicyType,normal),(d.Status,approved)}
t28	s10:acdih	{(a.Amount,595),(a.PolicyType,normal),(d.Status,approved)}
t29	s11:abdih	{(a.Amount,550),(a.PolicyType,normal),(d.Status,rejected)}
t30	s11:abdih	{(a.Amount,545),(a.PolicyType,normal),(d.Status,rejected)}
t31	s11:abdih	{(a.Amount,540),(a.PolicyType,normal),(d.Status,rejected)}
t32	s11:abdih	{(a.Amount,535),(a.PolicyType,normal),(d.Status,rejected)}
t33	s11:abdih	{(a.Amount,530),(a.PolicyType,normal),(d.Status,rejected)}
t34	s11:abdih	{(a.Amount,525),(a.PolicyType,normal),(d.Status,rejected)}
t35	s11:abdih	{(a.Amount,520),(a.PolicyType,normal),(d.Status,rejected)}
t36	s11:abdih	{(a.Amount,501),(a.PolicyType,normal),(d.Status,rejected)}
t37	s12:afih	{(a.Amount,641),(a.PolicyType,normal)}
t38	s13:ah	{(a.Amount,520),(a.PolicyType,normal)}
t39	s14:aeg	{(a.Amount,580),(a.PolicyType,normal)}
t40	s14:aeg	{(a.Amount,700),(a.PolicyType,normal)}
t41	s15:adfemh	{(a.Amount,1000),(a.PolicyType,normal),(d.Status,rejected)}
t42	s16:acdfmenlgh	{(a.Amount,0),(a.PolicyType,normal),(d.Status,rejected)}
t43	s16:acdfmenlgh	{(Amount,0),(PolicyType,normal),(Status,rejected)}

Fig. 4: Example log for a claim handling process.

been evaluated (task **d**, Evaluate claim), either an approval letter (task **e**, Send approval letter) or a rejection letter (task **i**, Send rejection letter) is sent to the customer. In the former case, a number of tasks are performed in order to eventually issue a payment for the

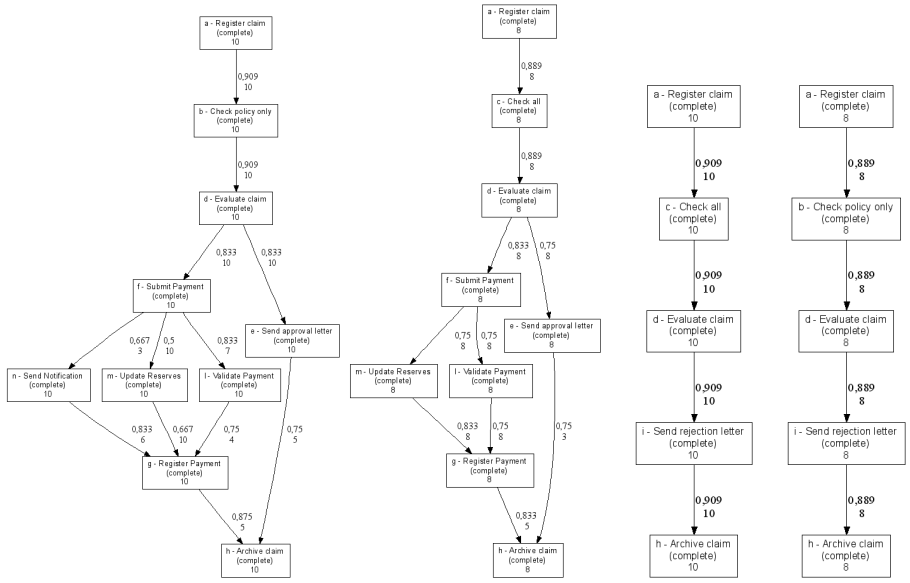
claim: task **f** (Submit Payment), task **l** (Validate Payment), task **m** (Update Reserves), task **n** (Send Notification), task **g** (Register Payment). Finally, the claim is archived and closed (task **h**, Archive claim). Notice that only the activities `a` and `d` have data items associated: the amount of money involved (*Amount*), the customer (*CustomerID*) and the type of policy (*PolicyType*) are all stored during claim registration (task `a`), while an annotation (*Status*) about claim acceptance/rejection is held after evaluating the claim (`d`). By the way, *Amount* is a numerical attribute, while both *PolicyType* and *Status* are nominal attributes taking values from $\{\text{“}normal''\text{,“}premium''\}$, and $\{\text{“}approved''\text{,“}rejected''\}$, respectively.

4.1 Discovery of Behavioral Clusters and Outliers

Let us first examine the behavior of algorithm `structuralClustering` against the example log of Figure 6, with $\sigma = 0.1$, $\gamma = 0.2$, $\alpha = 0.4$, and $\beta = 0.1$.

By applying function `FindPatterns`, a number of frequent structural patterns are found, which include those evidenced in Figure 6. By subsequently applying function `FindCoClusters` algorithm `structuralClustering` eventually discovers four different structural clusters: one with the traces $t_1, ...t_{10}$ (corresponding to the sequences $s_1, ..., s_5$ of Figure 6), one with the traces $t_{11}, ..., t_{18}$ (corresponding to s_6 and s_9), one with the traces $t_{19}, ..., t_{28}$ (all corresponding to sequence s_9), an the last with the traces $t_{29}, ..., t_{36}$ (corresponding to s_{10}). On the other hand, all of the remaining log traces (associated with $s_{11}, ..., s_{16}$) are recognized as anomalous process instances. It is worth noting that this is in line with the observations made in Example 3, concerning desirable outcomes of such a clustering process. Figure 5 shows some workflow models that we obtained by processing these clusters with a classical workflow discovery tool (namely the *HeuristicMiner* plugin ([28] provided by popular process mining framework ProM [25]), in order to get further hints on the results found by algorithm `structuralClustering`.

Despite the simplicity of the process and log considered here, these workflow schemas actually represent four major execution scenarios for the process itself, which mainly differ for the kind of policy check performed (Check policy only) vs. Check all)) and for the final decision (approval vs. rejection) on the claim. In general, such an effect can well help improve the precision of classical process mining approaches, by preventing the risk of having a single workflow that mixes up heterogeneous behaviors and models situations that do not happen in reality. This is, in fact, the case of the overall schema shown in Figure 6, which was obtained by directly applying such a process mining algorithm (plugin *HeuristicMiner* was again used to this end) to the whole log of Figure 4). Beside modelling some additional spurious task links (due to the presence of outlier traces $t37, ..., t41$), this latter workflow schema incorrectly allows, indeed, for simultaneously executing the tasks `e` (Send accept letter) and `i` (Send rejection letter), despite they only occur together in two (anomalous) log traces. Moreover, it does not capture the fact that task `n` (Send Notification) never occurred in the cases where a complete check



(a) Cluster 0 (b) Cluster 1 (c) Cluster 2 (d) Cluster 3

Fig. 5: Workflow schemas found by algorithm `structuralClustering` on the running example.

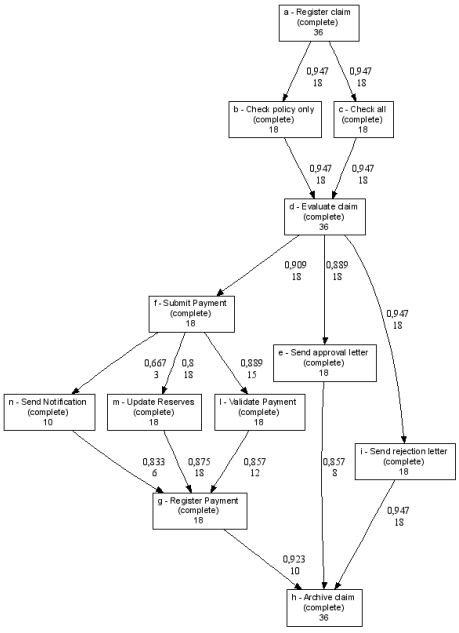


Fig. 6: Workflow schema found with a classical technique on the running example.

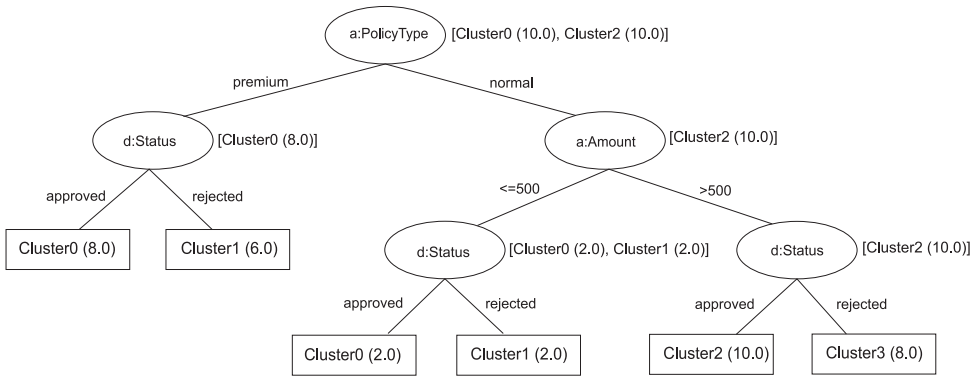


Fig. 7: A 0-compliant *DADT* found by algorithm `DADT-Induction` on the running example ($\omega = 0.35$).

of the claim was accomplished, by way of task `c` (Check all). It is worth noting that these behavioral rules, effectively captured via our clustering-oriented approach, correspond to very complicated workflow patterns (involving non-free choices and hidden tasks) that are beyond the scope of most process mining approaches. By the way, we also pinpoint that erroneous log traces like $t42$ and $t43$ would hardly be detected out by previous methods in the literature dealing with noised logs, which do not take account for the existence of different execution scenarios (i.e. trace clusters). As a matter of fact, since such methods focus on the frequency of tasks and of task links, they cannot recognize anomalous situations corresponding to the enactment of multiple process patterns that are frequent in the log separately, but rarely occur together with each other.

4.2 Discovery of Predictive Models

Let us now turn to the application of algorithm `DADT-Induction` to the 4 main clusters found by algorithm `structuralClustering`, in order to find a predictive model expressing the correlation of these behavioral classes with non structural process attributes. To this purpose, we retained all data attributes but *CustomerID* (which is indeed useless for learning general behavior) and set $minCardinality = 0$, $\sigma = 0.05$ and $\omega = 0.35$.

Figure 7 sketches the structure of the discovered *DADT* whereas Figure 8 reports the model returned when using the same setting for all the parameters but $\omega = 1$ — which practically corresponds to applying a classical decision-tree induction algorithm like C4.5 [18].

Differently from this latter tree, the topology of the model in Figure 7 fits well the task precedences expressed by the schemas of Figure 5. Interestingly, in this special case, this result has been achieved without paying any loss in the accuracy of the model (w.r.t. the input log) — which is indeed maximal for both trees in Figures 7 and 8.

A notable feature of the decision tree in Figure 7 is that each of its nodes is associated with a joint probability function, relating the node with each cluster. For the sake

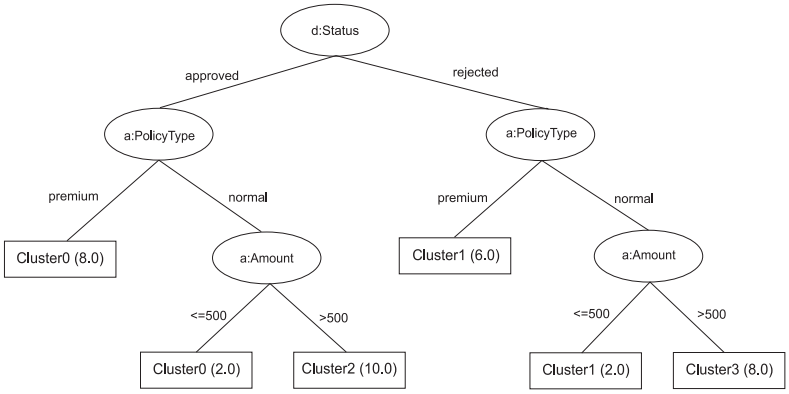


Fig. 8: A decision tree found on the running example without considering temporal aspects (i.e. $\omega = 1$)].

of compactness, we here only report, beside each node non-leaf node v, the most probable clusters for v and the number of input log traces that felt in v during the learning process. Clearly, such information can be exploited to predict cluster membership for ongoing process instances. For example, one can exploit the tree in Figure 7 to forecast that the uncompleted trace [<Register Claim,{(Amount,50),(PolicyType,premium)}>] will fall in *Cluster 0*. Conversely, the trace [<Register Claim,{(Amount,300),(PolicyType,normal)}>,<Check policy only,{}>] is estimated to eventually fall in either *Cluster 0* or *Cluster 1*.

5 Experiments

The approach proposed in the paper has been implemented integrated into a Java prototype system, which is meant to support the analysis workflow logs represented in the MXML format [26], used in the process mining framework ProM [25]. In particular, the system can be exploited to detect structurally homogeneous trace clusters and anomalous traces in a given input log. Moreover, given a set of logs, which are assumed to correspond to different classes of behaviors, the system allows to discover a decision tree model that predicts class membership based on context data encoded in the log traces —incidentally, MXML format already admits registering such kind of data, in the form of attribute-value pairs, which can be associated indeed with both process instances and low-level log events.

This section discusses the application of the proposed approach on two different real-life scenarios, with the aim of providing evidence for the practical usefulness of our proposal. The remainder of this section is organized as follows. We first introduce, in Section 5.1, a series of metrics enabling for a quantitative evaluation of experimental findings. Concrete datasets used in the experimentation are then illustrated in section 5.2. Section 5.3 offers a summarized view over a series of experiments and discusses results

obtained on overall quality metrics. A further series of test results, discussed in section 5.5, is finally aimed at evaluating the capability of the our decision tree models to support "on-the-fly" prediction over uncompleted process instance, compared with a classical decision-tree induction approach.

Observe that, in all of the experiments described next, a fixed setting for parameters σ, γ, α and β was considered: $\sigma = 0.05$, $\gamma = 4$, $\alpha = 0.4$, $\beta = 0.1$. In general, the tuning of these parameters is a tricky task in practical applications, which depends on the actual distribution of data. Yet knowing that an absolutely optimal setting can hardly be stated a-priori, our choice was mainly founded on observing that these values ensured satisfactory results in several tests we carried out on different kinds of synthesized data (with the aim of studying the sensibility of the approach to its parameters). Details on these latter tests have been omitted from this paper for space reasons, as we believe that such an issue is beyond the scope of this paper. On the contrary, different values of coefficient ω will be explored, in order to allow the reader appreciate the advantage of using our time-oriented approach to discovering a decision tree models for execution scenarios' prediction.

5.1 Evaluation setting

In the evaluation of experiment results we focused on two main aspects: *(i)* the quality of discovered workflow models, as concerns specifically their ability to precisely model the structure of process instances, by possibly capturing different execution scenarios; and *(ii)* the quality of discovered *DADT* decision trees, as concerns their capability to predict the structural class of process instances based on non-structural information, and to fit temporal aspects of the process. A number of metrics adopted to this purpose are illustrated next, which allow to quantitatively evaluate the quality of both kinds of models.

Quality of structural (workflow-based) models. The conformance of a workflow model W w.r.t. a log L can be measured through three complementary metrics (all defined in [20]), ranging each over real interval [0,1]:

- *Fitness*, which essentially evaluates the ability of W to parse all the traces L, by indicating how much the events in L comply with W.
- *Advanced Behavioral Appropriateness* (denoted by *BehAppr*, for short), which estimates the level of flexibility allowed in W (i.e., alternative/parallel behavior) really used to produce L.
- *Advanced Structural Appropriateness* (or *StrAppr*, for short), which assesses the capability of W to describe L in a maximally concise way.

These measures were defined for a workflow schema and cannot apply directly to the *MSSM* model discovered by our approach. In order to have a single overall score for such

a model, we simply average the values computed by each of these measures for each workflow schema (w.r.t. its associated trace cluster). More precisely, the conformance values of these schemas are added up in a weighted way, where the weight of each schema is the fraction of original log traces that constitute the cluster it was mined from.

Quality of predictive (*DADT*) models. For evaluating the precision of *DADT* models we essentially resort to the classical *Accuracy* measure, expressing the percentage of correct predictions that would be made over all possible traces of the process, estimated with 10-fold cross-validation [15]. By the way, we will also compute this measure against incomplete log traces in order to assess the capability of *DADT* models to carry out "on-the-fly" predictions (see section 5.5). In addition, in order to provide a "local" evaluation of classification accuracy, we report, for each single cluster c (i.e. behavioral class), standard measures of *precision* (P_c), *Recall* (R_c) and the well-known *F-measure* ($\mathcal{F}_c = ((\beta^2+1)P_c \times R_c)/(P_c+\beta^2 R_c)$, coinciding with the harmonic mean of the precision and recall values for $\beta = 1$).

As a further quality measure, an overall score (ranging in real interval [0,1]) is introduced which indicates as much the model complies with the precedence relationships among process tasks. In order to make such an evaluation independent of discovered workflow models, we only base compute it against the log, by measuring, for each leaf node l and for each trace t assigned to l, as much the ordering of tasks within t agrees with the sequence of split tests that lead from the root to l. More formally:

Definition 8. Let L be a log over task set T and attribute set A, and D be a decision tree over the same attribute set. For any leaf l of D, let *(i)* $a_1^l \ldots a_k^l$ be the attributes associated with the sequence of non-leaf nodes $n_1^l \ldots n_k^l$ in the path from $\mathcal{D}$'s root to l —i.e. $a_i^l = \mathcal{D}.\texttt{attr}\,(n_i^l)$, for $i = 1..k$—, and *(ii)* $path(l) = p_1^l, \ldots, p_k^l$ be the sequence of tasks corresponding to $a_1^l \ldots a_k^l$ —i.e. $p_i^l = task(a_i^l)$ for $i = 1..k$. Then, the *conformance of $\mathcal{D}$ w.r.t. L*, denoted by $Conf(\mathcal{D},L)$ is defined as follows:

$$Conf(\mathcal{D},L) = \frac{1}{N} \sum_{\text{leaf } l \text{ of } \mathcal{D}} \sum_{t \in l} \left(1 - \frac{mismatches(t, path(l))}{maxMismatches(t, path(l))}\right)$$

where $mismatches(t, path(l))$ is the number of times the task precedences in t are inverted in $path(l)$, while $maxMismatches(t, p_l) = \frac{|t \cap p_l||t \cap p_l - 1|}{2}$ is the maximum number of such inversions that may occur between two sequences containing the same tasks as t and $path(l)$, respectively. Moreover, for any *DADT* model $\mathcal{D} = \langle D, \texttt{attr}, split, p \rangle$, we will also denote $Conf(\mathcal{D},L) = Conf(\mathcal{D}.D, L)$. □

Example 4. Consider the decision trees in Figures 7 and 8, and the example log in Figure 4. Let l_1^a and l_1^b indicate the leftmost leaf in the tree of Figure 8 and of Figure 7, respectively. Let us also denote by t_1 the first trace in the log of Figure 4, which clearly corresponds to the task sequence `abdflenmgh` . When considering the first tree, t_1 is clearly

assigned to l_1^a, which is associated with the task sequence $path(l_1^a) = ad$. Conversely, in the other tree, t_1 is assigned to l_1^a, which is corresponds to the task sequence $path(l_1^a) = da$. Therefore, as concerns the classification of trace t_1, the first tree (induced with $\omega = 1$) causes 1 mismatch, while no mismatch arises with the second tree (mined using $\omega = 0.35$)— indeed, it is $mismatches(t_1, path(l_1^a)) = 1$, and $mismatches(t_1, path(l_1^b)) = 0$.

It is worth noting that the measure $Conf(\mathcal{D}, L)$ defined above is a pessimistic estimation for the capability of a *DADT* $\mathcal{D}$ to comply with the workflow models that could be discovered for the log L, by using some suitable process mining technique. In fact, if, e.g., d and b are parallel activities, the log L is likely to contain both some trace t_{db} where d precedes b and some trace t_{bd} where conversely b occurs before d. Then, for any *DADT* $\mathcal{D}$ that make use both tasks, the $Conf(\mathcal{D}, L)$ will incorrectly count a mismatch on either t_{db} or t_{bd}.

5.2 Datasets

Experimental activities were carried out on datasets coming from two different real-life application scenarios, which are described in the two following subsections, respectively.

Data from a logistic system (Logs *A* and *B*). The first application scenario concerns the operational system used in an Italian maritime container terminal. Basically, the life cycle of any container is roughly summarized as follows. The container is unloaded from the ship and temporarily placed near to the dock, until it is carried to some suitable yard slot for being stocked. Symmetrically, at boarding time, the container is first placed in a yard area close to the dock, and then loaded on the cargo. Different kinds of vehicles can be used to move a container, including, e.g., cranes, straddle-carriers (a vehicle capable of picking and carrying a container, by possibly lifting it up), and multi-trailers (a sort of train-like vehicle that can transport many containers). Each container hence undergoes different logistic operations which determine its displacement across the "yard", i.e., the main area used in the harbor for storage purposes, logically partitioned into bi-dimensional *slots*. Slots are the units of storage space used for containers, and are organized into disjoint *sectors*.

In our experimentation, we focused on a subset of 5389 containers, namely the ones that completed their entire life cycle in the hub along the first two months of year 2007, and which were exchanged with four given ports around the Mediterranean sea. In order translate these data into a process-oriented form, we regarded the transit of any container through the hub as a single enactment case of a (unknown) logistic process, and derived the following logs, based on two different analysis perspectives:

- ***Log A*** ("operation-centric"), storing the sequence of basic logistic operations applied to the containers. More precisely, the following distinct operations may be registered

for any container c: MOV (c was moved from a yard position to another by a straddle carrier), DRB , (c was moved by a multi-trailer), DRG (a multi-trailer movedd to get c), LOAD (c was charged on a multi-trailer)), DIS (c was discharged off a multi-trailer), SHF (c was moved upward or downward, possibly to switch its position with another container), OUT (a dock crane embarked c onto a ship).

– ***Log B*** ("position-centric"), registering the flow of containers across the yard. Here, the focus is on the slot/sector associated with each logistic event.

In both cases, two dummy activities, denoted by START and END , were introduced to univocally mark the beginning and the end of each log trace, respectively. Further, various data attributes have been considered for each container (i.e., for each process instance), including its origin and final destination ports, its previous and next calls, diverse characteristics of the ship that unloaded it, its physical features (e.g., size, weight), and a series of categorical attributes concerning its contents (e.g., the presence of dangerous or perishable goods). All of these data have been encoded in both logs as attributes of task START .

Data from a collaboration work platform (Log *CAD*). This second application scenario, studied in the research project *TOCAI.it* [2], concerns the collaborative processes performed in a manufacturing enterprise in order to carry out the design and prototypical production and test of new items (i.e., both final artifacts and components).

In this scenario, the design of a new item is accomplished by handling one or more CAD projects through a distributed CAD platform, which allows different kinds of actors to work in a cooperative and concurrent way.

Precisely, the following kind of events can be traced for each project: Creation , Construction (start of design for the item associated with the project), Modify (the project was saved and a new version of it started off), CancelModify (the last modification to the project was undone), Musterbau (a prototype was built for an item), Pruefung (the project was validated), TechAend (a technical revision was done for an item), Share (the project was shared with other workers), Release (the project was released), NullSerie (a pilot series was produced).

In particular, we focused on the operations performed, in the first three months of year 2007, over 5794 projects that were never renamed —i.e. having just one single occurrence of operation Creation .

These historical were restructured into a process log, referred to as *Log CAD* hereinafter, where each log trace corresponds to a distinct project, and records the sequence of CAD operations performed on the project. Each operation occurrence was also associated, in the log, with two attributes, concerning the user that performed it: the working

[2] TOCAI.it (Tecnologie Orientate alla Conoscenza per Aggregazioni di Imprese in Internet), research project funded by Italian Ministry of University and Scientific Research.

group he/she belonged to (*Group*), and the role he/she was playing within the design process (*Role*).

5.3 Summary of Experiment Results

Tables 5.3 and 5.3 summarize the outcomes of a selection of experiments performed on the log data described above. In particular, the former table reports the number of clusters found by algorithm *A-ESD*, and the quality scores computed for the *MSSM* eventually returned as output. Table 5.3 shows instead some important figures of the *DADT* models obtained with algorithm *A-ESP*, just based on the clusters and structural models found by *A-ESD*. In particular, for each such classification model, its size and accuracy are reported, as well as its conformance to the input log, measured according to the $Conf$ measure defined in Section 5.1.

Table 1: Summary of results algorithm *A-ESD*.

Log	**Clusters**	**Fitness**	**BehAppr**	**StrAppr**
Log *A*	2	0.8725	0.9024	1.0
Log *B*	5	0.8558	0.9140	1.0
Log *CAD*	4	0.6842	0.6584	1.0

Table 2: Summary of results of algorithm *A-ESP*.

Test			**Clusters**	**Data-aware classification model**		
Dataset	*Attributes*	ω		*Accuracy*	*Tree Size*	*Conf*
Log *A*	*case*	1	2	96.01%	69	1.0
	all	1	2	98.03%	147	1.0
	all	0.6	2	97.49%	101	1.0
Log *B*	*case*	1	5	91.64%	105	1.0
	all	1	5	94.98%	135	0.89
	all	0.6	5	95.01%	135	0.98
Log *CAD*	*task*	1	4	71.62%	19	0.49
	task	0.6	4	72.47%	45	0.72

Different settings were considered for the application of algorithm *A-ESP*, which differ for the value of parameter ω (while keeping fixed $\sigma = 0.15$, $\alpha = 0.6$), and for the kind of non-structural information considered: only case attributes ($Attributes = case$), only task attributes ($Attributes = task$), or all of them ($Attribute = all$). In this regard, we observe that in the case of Log *CAD*, all the attributes available —namely, the group and role of users performing each single CAD operation— refer to task elements, and there are no case attributes.

In particular, as concerns the setting of parameter ω, we here only focus on two different options:

1. $\omega = 1$, which practically makes our approach coincide with the J48 algorithm — indeed, in this case all precedence constraints in the structural models are completely ignored when inducing the decision tree model—, and
2. $\omega = 0.6$, where conversely a *DADT* model is built by taking into account such information, based on the algorithmic scheme shown in Figure 3.

The value 0.6 was chosen in a pragmatical way, based on the observation that it ensured a good compromise between classification accuracy and structural conformance. However, similar results were obtained when using different values of ω, in the real range $(0.3, 0.7)$.

In general, the results shown in tables 5.3 and 5.3 confirm that the proposed approach allowed to achieve good effectiveness results in all the considered analysis scenarios, as concerns the modelling of both structural and non-structural aspects of of the logged events. It is also interesting to observe that this precision does not come with a verbose (and possibly overfitting) representation. Indeed, for all the tests, the number of clusters and the size of the tree are quite restrained, while the workflow models collectively attain a maximal score with the *StrAppr* metric.

Table 3: Top level attributes in the *DADT* models.

Test			**Top Level Attributes**
Dataset	*Attributes*	ω	
`Log` *A*	*case+task*	0.6	PrevHarbor, ShipType_OUT, NavLine_IN, ContType
`Log` *B*	*case+task*	0.6	ShipSize_IN, ShipType_OUT, ContType, PrevHarbor
`Log` *CAD*	*task*	0.6	Creation::Group, Creation::Role, Construction::Group, Share::Group

As a complement to these results, Table 3 reports 4 top-ranked attributes for each dataset, i.e. the 4 attributes that most frequently appeared in the the top levels of the decision trees discovered from each dataset.

As mentioned above, by contrasting the results obtained with $\omega = 0.6$ to those obtained with $\omega = 1$, we can have a sort of comparison between the induction technique introduced in 3.2 (and sketched in Figure 3) with classical decision-tree induction algorithms, such as C4.5 and its variant J48 [8, 18]. In this regard, we first notice that such analysis degenerates in the case of log A, where the non-structural information relevant to discriminating the two structural clusters is conveyed by case attributes, with just one of task attribute (namely the distance covered in the first `MOV` operation) playing a marginal role. As a consequence, even when task precedences are ignored in the induction of the classification model ($\omega = 1$), a maximal conformance value is obtained for this model.

Conversely, perturbing the attribute selection criterion with our heuristic based on task precedences produces a slight decrease in the accuracy of the model, mainly due to the fact that additional constraints limit the selection of most predictive features.

Such an effect does not arises on the logs *B* and *CAD*, where, as expected, our technique allows to improve the conformance of the classification model. Interestingly, in these cases, the capability of the decision tree to predict the behavior of log traces is improved when using the precedence-based heuristic in the selection of split attributes ($\omega = 0.6$). Such a beneficial effect was completely unexpected, and seems to suggest that in some case considering the logics of the business process can guarantee better results than inducing the classification model via the classical greedy approach, based on entropy reduction.

5.4 Detailed results: some of discovered models

We next focus on the findings of some tests performed on each of the three dataset presented so far.

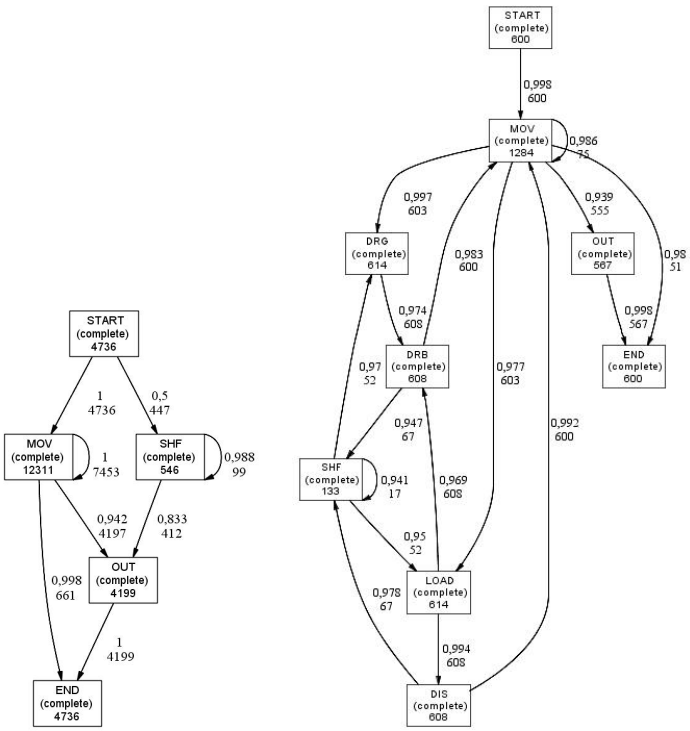
Log A. As specified above, each trace of *Log A* encodes the sequence of basic operations (i.e., `MOV`, `DRB`, `DRG`, `LOAD`, `DIS`, `SHF`, `OUT`) applied to each single container. In addition to container attributes, a series of data attributes were associated with each occurrence of these operations, including the human that carried out the operation (*Originator*), the two positions the container was moved between (*FromPosition* and *ToPosition*, resp.), the completion time (*Timestamp*) and the duration of the operation (*ElapsedTime*), the distance covered (*Distance*) and the kind of vehicle used in the operation (*Vehicle*). As an example, we next discuss the results obtained when applying our approach to the log described above with

As an example, we next discuss the results obtained when applying our approach to the log described above with $\omega = 0.6$, considering all data attributes for *DADT* induction.

In this case, algorithm *A-ESD* discovered two distinct normal usage scenarios, and 53 outlier traces. Subsequent analyses by domain experts confirmed that most of outlier individuals (i.e. container histories) actually correspond to anomalous cases and to malfunctions in the tracking system. Further details are omitted for privacy restrictions.

Structural aspects of the scenarios are described by the workflow schemas shown in Figure 9, which essentially differ for the presence of operations performed with multi-trailer vehicles: the schema of Figure 9.(a)) does not feature any of these operations, which are instead contained in the other schema. Notably, the former schema captures the vast majority of handling cases (4736 containers of the original 5389 ones). This reflects a major aim of yard allocation strategies: to keep each container as near as possible to its positions of disembarkation/embark, by performing short transfers via straddle-carriers.

Interestingly, high quality scores were obtained by these structural models over all of the conformance measures: $Fitness = 0.8791$, $BehAppr = 0.9089$, $StrAppr = 1.0$.



(a) Cluster 0 (b) Cluster 1

Fig. 9: Results on log A ($\omega = 0.6$ and $\sigma = 0.1$): the two workflow schemas found.

Moreover, an astonishing 97.49% accuracy score was achieved by the data-aware classification model discovered for the log, so confirming that these two markedly different execution scenarios strongly depend on process features that go beyond the mere sequencing of yard operations. Among these features, the following container properties stood out: the provenance port of a container (`PrevHarbor `), the kind of ship that that is going to take it away (`ShipType _OUT`), the navigation line delivering the container to the hub (`NavLine _IN`) and the kind of container (`ContType `) (e.g., fridge container). As to the attributes of tasks (i.e., operations), we notice that only those associated with the `MOV` operation are actually used by the classification model.

For space and privacy reasons, we do not show the decision tree model found in the test. However, in order to provide some hint of the practical usefulness of such a model, in Figure 10 we report two of its branches, which were deemed quite useful for explaining and discriminating the discovered usage scenarios. Both branches are represented as if-then rules, r_1 and r_2, while the notation *MOV::Distance* is adopted to denote the attribute *Distance* associated with the task *MOV*. Notably, r_1 is a very selective rule clas-

r_1:
if PrevHarbor = ANR
and ShipType_OUT ≠ RR
and ShipType_OUT ≠ CF
and ContType = DC
and MOV::Distance ≤ 204
then *Cluster 0*

r_2:
if PrevHarbor = ANR
and ShipType_OUT ≠ RR
and ShipType_OUT ≠ CF
and ContType = DC
and 204 < MOV::Distance ≤ 354
then *Cluster 1*

Fig. 10: Results on log A: an excerpt of the decision tree.

sifying just 11 traces of the overall 5336 as belonging to *Cluster 0* with an accuracy of 81.82%. Conversely, rule r_2 assigns 52 traces of the input log to *Cluster 1* and gets 94.23% accuracy.

A finer grain analysis, conducted with the help of Table 4 (where individual precision/recall measures for the two clusters are shown), confirms that the model guarantees a high rate of correct predictions for either cluster.

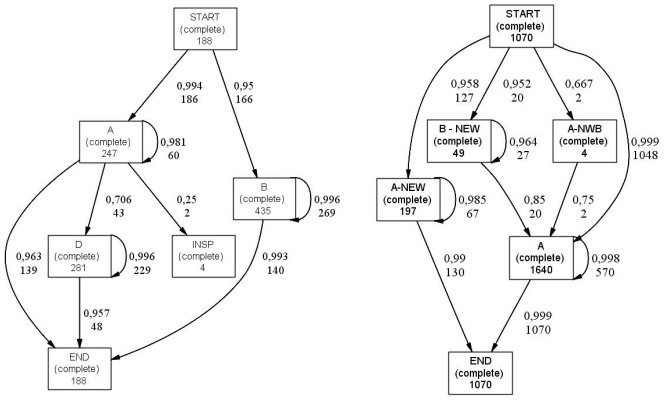
Table 4: Results on log A ($\omega = 0.6$ and $\sigma = 0.1$): details on the discovered clusters.

Cluster	*Size*	*P*	*R*	$\mathcal{F}(\beta = 1)$
0	4736	98,24%	98,94%	98,59%
1	600	91,17%	86,00%	88,51%

Log B. We recall that this log was created to arrange original data into a "position-centric" fashion, in order to capture the paths typically followed by the containers around the yard. Precisely, each trace in *Log B* encodes the sequence of yard sectors occupied by a single container during its stay. Each log event is also associated with several non-structural data attributes, which include the human who moved the container (*Originator*), the distance covered (*Distance*), the time spent to move the container (*Elapsed-Time*), the kind of vehicle used (*Vehicle*), and the working turn during which it happened (*Turn*).

We next focus on the results obtained in one of the experiments we carried out against this process log, where the parameter ω was set again to 0.6, and all data attributes were taken into account for inducing the *DADT* model.

The structural clustering performed by algorithm *A-ESD* allowed to recognize 5 trace clusters, corresponding to prevalent behaviors, and 63 outlier traces. We remark that, in principle, due to the high number of sectors and moving patterns that come to play in such analysis perspective, any flat representation of container flows, just consisting of a single workflow schema, risks being either inaccurate or difficult to interpret. Conversely, by separating different behavioral classes our approach ensures a modular representation, which can better support explorative analyses. In fact, the five clusters found



(a) Cluster 1 (b) Cluster 3

Fig. 11: Results on log *B*: two workflow schemas.

in this test have been equipped with clear and compact workflow schemas, which exhibited high levels of conformance with the log: $Fitness = 0.8687$, $BehApp = 0.9254$, $StrAppr = 1.0$. As instance, two of these schemas are shown in detail in Figure 11, which differ both in the usage of sectors and in some of the paths followed by the containers across these sectors.

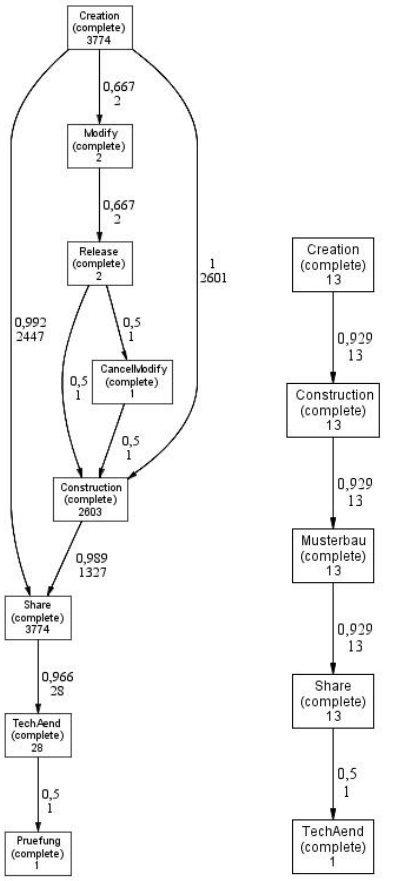
Interestingly, a satisfactory accuracy 95.01% is achieved again by the *DADT* model. As a matter of fact, by comparing these results with those obtained in the previous test, we notice that a slightly lower precision and a larger size of the decision tree (cf. Table 5.3), mainly due to the higher level of complexity that distinguish the position-centric analysis from the operation-centric one. Incidentally, Table 5 reveals that such worsening is mainly to blame on the inability of *DADT* to appropriately recognize well the *Cluster 1*, which is, in fact, slightly confused with the *Cluster 3*.

Table 5: Results on log *B* ($\omega = 0.6$ and $\sigma = 0.1$): details on the discovered clusters.

Cluster	*Size*	*P*	*R*	$\mathcal{F}(\beta = 1)$
0	3660	96,84%	98,55%	97,69%
1	187	94,57%	64,89%	76,97%
2	344	95,00%	93,14%	94,06%
3	1068	88,99%	92,15%	90,54%
4	67	94,29%	97,06%	95,65%

We finally notice that almost the same attributes as in the former test have been employed to discriminate the clusters (even though in a different order), except for the usage

of `ShipSize _IN` (i.e., the size category of the ship that delivered the container) in place of `NavLine _IN`.



(a) Cluster 0 (b) Cluster 3

Fig. 12: Results on log *CAD*: two workflow schemas.

Log CAD. In the remainder of this subsection, we focus on one experiment conducted over this log, where the proposed approach was used with $\omega = 0.6$ and by considering all data attributes.

Table 6: Results on log *CAD* ($\omega = 0.6$ and $\sigma = 0.1$): details on the discovered clusters.

Cluster	*Size*	*P*	*R*	$\mathcal{F}(\beta = 1)$
0	3774	78,13%	87,85%	82,71%
1	825	56,99%	19,83%	29,42%
2	1138	56,08%	59,58%	57,78%
3	13	100%	69,23%	81,82%

Four different clusters and associated workflow models were discovered in this experiment, which capture the behavior registered in the log in an adequate enough manner — the global conformance scores of the structural model are, indeed, $Fitness = 0.6933$, $BehApp = 0.6687$, $StrAppr = 1.0$. By the way, 50 traces were perceived as outliers, which actually corresponds to unusual developments of CAD projects.

A *DADT* classification model was also discovered, consisting of 45 nodes, which achieves 72.47% prediction accuracy, based on the information about the role and group of the users that performed some of the CAD operations —primarily, `Creation` , `Construction` and `Share` . Despite a lower precision score is achieved than in the previous application scenario, this result is quite surprising, as there was no a-priori expectation that users' roles and groups could be correlated with different CAD scenarios, and could really help discriminate among them.

Figure 12 shows the workflow models discovered for two (of four) clusters found. In particular, it is easy to see that *Cluster 3* correspond to a somewhat anomalous execution case —which actually regards only 13 traces— where a project is built but it is never validated. Interestingly, the *DADT* classification model discovered in the experiment, is able to precisely predict even this outsider behavior (see Table 6).

In Figure 13, we show four rules extracted from *DADT* model, all of which are quite interesting and accurate. Specifically, r_3 is a simple and yet very precise two-level rule which only captures 6 traces of *Cluster 1*, with 100% accuracy. This result even more interesting if we consider the disappointing performances the tree has in predicting this cluster (cf. Table 6). 592 traces of the input log are assigned instead to *Cluster 0* by rule r_4, which also achieves maximal precision (100% accuracy). Rule r_5 gets some lower precision result —it classifies 339 traces with 66.05% of accuracy— but it is still helpful, in that it evidences the compliance of the tree w.r.t. to schemas shown in Figure 12. Finally, rule r_5, which correctly assigns 11 log traces to *Cluster 0*, demonstrates that satisfactory results (an accuracy of 92.46%) can be achieved despite of the constraints imposed by precedence relationships of the workflow schemas.

5.5 Estimating on-the-fly prediction power

A further kind of experiment was performed to assess the advantage of using our decision tree induction technique within an "on-the-fly" prediction setting, such as the one

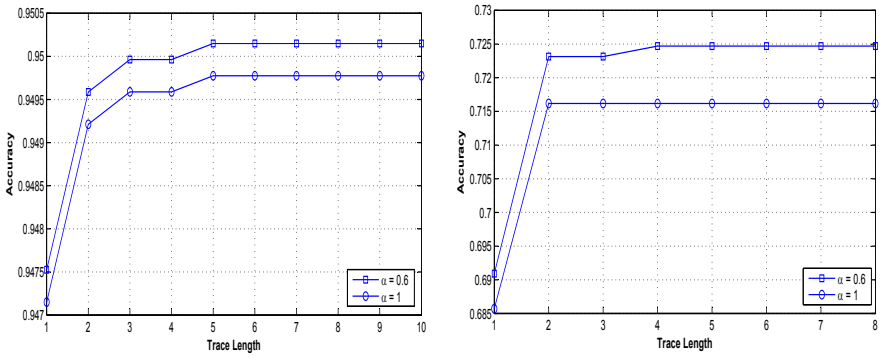
r_3: **if** Creation::Role = User **and** Creation::Group = EKM **then** *Cluster 1*

r_4: **if** Creation::Role = User **and** Creation::Group = EKS_J **then** *Cluster 0*

r_5: **if** Creation::Role = User **and** Creation::Group = EK_H **and** Construction::Group = EK_H **then** *Cluster 2*

r_6: **if** Creation::Role ≠ User **and** Creation::Group = CAD_M **and** Share::Group ≠ EKD_J **then** *Cluster 0*

Fig. 13: Results on log *CAD* ($\omega = 0.6$ and $\sigma = 0.1$): an excerpt of the decision tree.



(a) Log *B* (b) Log *CAD*

Fig. 14: "On-the-fly" prediction on real data: classification accuracy vs. trace fragments' length.

discussed in Section 1, where the behavioral cluster of a forthcoming process instance should be estimated possibly before it has been completed. In order simulate such a kind of analysis, we measured the accuracy of the classification model over datasets, each of which contains a k-prefix of each log traces, for k ranging from 1 to the maximal trace length.

For two of the real-life logs, namely *LogA* and *LogB*, Figure 14 depicts the accuracy of the classification model, in correspondence of each of these log subsets (i.e., for different trace lengths). Two plots are shown for each log: one for the decision tree discovered by using the algorithm in Figure 3 with $\omega = 0.6$, and one for the decision tree found with *J*48 —this practically corresponds to set $\omega = 1$ in our prototype system.

Clearly, in all cases, the classification models make better predictions over longer traces. However, it is encouraging to notice that, in both application scenarios, our technique always guarantees higher accuracy results than the classical decision-tree induction method.

References

1. A. Apostolico, M. E. Bock, S. Lonardi, and X. Xu. Efficient detection of unusual words. *Journal of Computational Biology*, 7(1/2):71–94, 2000.
2. S. Basta, F. Folino, A. Gualtieri, M. A. Mastratisi, and L. Pontieri. A knowledge-based framework for supporting and analysing loosely structured collaborative processes. In *ADBIS (local proceedings)*, pages 140–153, 2008.
3. R. P. Jagadeesh Chandra Bose and Wil M. P. van der Aalst. Context aware trace clustering: Towards improving process mining results. In *Proc of the SIAM International Conference on Data Mining (SDM 2009)*, pages 401–412, 2009.
4. M. Burgess. Probabilistic anomaly detection in distributed computer networks. *Sci. Comput. Program.*, 60(1):1–26, 2006.
5. I. S. Dhillon, S. Mallela, and D. S. Modha. Information-theoretic co-clustering. pages 89–98, 2003.
6. A. J. Enright, S. Van Dongen, and C. A. Ouzounis. An efficient algorithm for large-scale detection of protein families. *Nucleic Acids Res*, 30(7):1575–1584, 2002.
7. T. E. Fawcett and F. Provost. Fraud detection. pages 726–731. Oxford University Press, 2002.
8. E. Frank, M. A. Hall, G. Holmes, R. Kirkby, and B. Pfahringer. Weka - a machine learning workbench for data mining. In *The Data Mining and Knowledge Discovery Handbook*, pages 1305–1314. 2005.
9. L. Ghionna, G. Greco, A. Guzzo, and L. Pontieri. Outlier detection techniques for process mining applications. In *Proc. of the 17th Intl Symposium on Foundations of Intelligent Systems (ISMIS 2008)*, pages 150–159, 2008.
10. G. Greco, A. Guzzo, and L. Pontieri. Mining taxonomies of process models. *Data & Knowledge Engineering*, 67(1):74–102, 2008.
11. G. Greco, A. Guzzo, L. Pontieri, and D. Saccà. Discovering expressive process models by clustering log traces. *IEEE Transactions on Knowledge and Data Engineering*, 18(8):1010–1027, 2006.
12. D. Grigori, F. Casati, U. Dayal, and M. Shan. Improving business process quality through exception understanding, prediction, and prevention. In *Proc. of 27th International Conference on Very Large Data Bases (VLDB'01)*, pages 159–168, 2001.
13. S. Jiang, X. Song, H. Wang, J. J. Han, and Q. H. Li. A clustering-based method for unsupervised intrusion detections. 27(7):802–810, 2006.
14. L. Maruster, A. J. M. M. Weijters, W. M. P. van der Aalst, and A. van den Bosch. A rule-based approach for process discovery: Dealing with noise and imbalance in process logs. *Data Mining and Knowledge Discovery*, (1):67–87, 2006.
15. T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
16. J. R. Quinlan. Discovering rules by induction from large collections of examples. In D. Michie, editor, *Expert systems in the micro electronic age*. Edinburgh Univ. Press, 1979.
17. J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.
18. J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
19. A. Rozinat and W. M. P. van der Aalst. Decision mining in ProM. In *Proc. of 4th Intl. Conf. on Business Process Management (BPM'06)*, pages 420–425, 2006.
20. A. Rozinat and W. M. P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95, 2008.
21. T. Hoffmann S. Dustdar and W. M. P. van der Aalst. Mining of ad-hoc business processes with teamlog.
22. Minseok Song, Christian W. Günther, and Wil M. P. van der Aalst. Trace clustering in process mining. In *Business Process Management Workshops*, pages 109–120, 2008.
23. S. Subramaniam, V. Kalogeraki, D. Gunopulos, F. Casati, M. Castellanos, U. Dayal, and M. Sayal. Improving process models by discovering decision points. *Information Systems*, 32(7):1037–1055, 2007.
24. W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters. Workflow mining: A survey of issues and approaches. *Data Knowledge Engineering*, 47(2):237–267, 2003.

25. B. F. van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and W. M. P. van der Aalst. The ProM framework: A new era in process mining tool support. In *Proc. of 26th International Conference on Applications and Theory of Petri Nets (ICATPN '05)*, pages 444–454, 2005.
26. B. F. van Dongen and W. M. P. van der Aalst. A meta model for process mining data. In *Proc. of EMOI-INTEROP*, pages 309–320, 2005.
27. Gabriel M. Veiga and Diogo R. Ferreira. Understanding spaghetti models with sequence clustering for ProM. In *Business Process Intelligence (BPI 2009): Workshop Proceedings*, 2009.
28. A. J. M. M. Weijters and W. M. P. van der Aalst. Rediscovering workflow models from event-based data using little thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003.