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

Mining Usage Scenarios in Business Processes: Outlier-Aware Discovery and Run-Time Prediction

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

RT-ICAR-CS-10-10 **Novembre 2010**

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

Mining Usage Scenarios in Business Processes: Outlier-Aware Discovery and Run-Time Prediction

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

Rapporto Tecnico N.: **RT-ICAR-CS-10-10**

Data: **Novembre 2010**

---

[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)

Mining Usage Scenarios in Business Processes: Outlier-Aware Discovery and Run-Time Prediction

Abstract

A prominent goal of process mining is to build automatically a model explaining all the episodes recorded in the log of some transactional system. Whenever the process to be mined is complex and highly-flexible, however, equipping all the traces with just one model might lead to mixing different usage scenarios, thereby resulting in a spaghetti-like process description. This is, in fact, often circumvented by preliminarily applying clustering methods on the process log in order to identify all its hidden variants. In this paper, two relevant problems that arise in the context of applying such methods are addressed, which have received little attention so far: (i) making the clustering aware of outlier traces, and (ii) finding predictive models for clustering results.

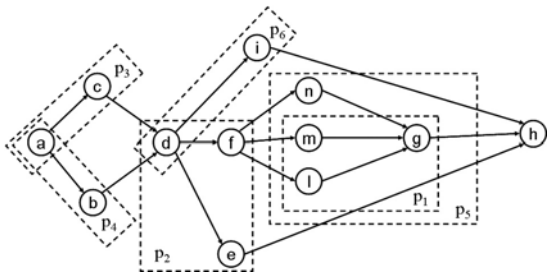
The first issue impacts on the effectiveness of clustering algorithms, which can indeed be led to confuse real process variants with exceptional behavior or malfunctions. The second issue instead concerns the opportunity of predicting the behavioral class of future process instances, by taking advantage of context-dependent "non-structural" data (e.g., activity executors, parameter values). The paper formalizes and analyzes these two issues and illustrates various mining algorithms to face them. All the algorithms have been implemented and integrated into a system prototype, which has been thoroughly validated over two real-life application scenarios.

Keywords: Business Processes, 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 [1]. 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 it. Eventually, the "mined" model can be used to design a detailed process schema capable of supporting forthcoming enactments, or to shed light on its actual behavior.

Traditional process mining approaches focus on capturing the "structure" of the process by discovering models that mainly express inter-task dependencies via precedence/causality links and other routing constructs specifying, for instance, the activa-



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)

Figure 1: A schema (left) and a log (right).

tion/synchronization of concurrent branches, exclusive choices, and loops over all the registered traces. As an example, given the event log (over tasks a, b, ...h) consisting of the traces shown on the right side of Figure 1 along with their associated frequency [1], a traditional process mining algorithm would derive an explicative model such as the one shown on the left, which represents a simplified process schema with no routing constructs, and where precedence relationships are depicted as directed arrows between tasks[2]. 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 mixing 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 [2–4] has affirmed the opportunity to recognize automatically different usage scenarios by clustering the input traces based on their behavioral/structural similarity. In particular, great effort has been spent on defining suitable metrics for measuring the similarity between log traces, which is a pre-requisite for clustering algorithms.

However, some technical and conceptual questions involved in the problem of clustering process traces have not been investigated so far, 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 occured in the enactments, clustering approaches for process mining have been proven to be effective in discovering accurate sets of process models. However, logs often reflect temporary malfunctions and anomalies in evolutions, whose understanding may help recognize critical points in the process potentially yielding invalid or inappropriate behavior. Indeed, if such exceptional individuals (referred to as *outliers* in the literature) are not properly identified, then clustering algorithms will likely mix the actual variants with specific behaviors that do not represent any usage scenario, but which rather reflect some malfunctioning in the system.

[1] E.g., the log contains 2 instances of s_1.

[2] E.g., m must be executed after f, while it can be executed concurrently with both l and n. Notice that, in addition to precedence links between task nodes, a number of subgraphs (labelled with $p_1, \ldots, p_5$), are depicted over the flow graph. Roughly speaking, this is the kind of core structural patterns exploited in our approach to discover both clusters and outliers in the given log traces, as discussed in detail in Section 3.

(2) Predictive Models. A tacit assumption in 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 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). If such additional non-structural information is available at run-time, a natural question then comes into play about whether it can be used to predict the cluster where the current enactment will likely belong to. In other words, one may ask for revealing the hidden associations between the cluster structure and the underlying non-structural data. Knowing these associations, in fact, paves the way for building forecasting tools (in the spirit of [5, 6]) predicting as accurately as possible the behavioral class of the current enactment.

Despite their relevance for practical applications, the problems of singling out outliers from the input traces and of finding predictive models for clustering results have received little attentions so far. The aim of this paper is 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, after reviewing relevant related works in Section 2, the problem of identifying anomalous traces in the context of process mining applications is faced in Section 3. To this end, an approach is proposed which characterizes the "normality" of a given set of traces, by mining structural patterns frequently occurring in the log and by clustering the traces based on their correlations with those patterns. Outliers are eventually reckoned 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 Section 4, 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 is addressed. The goal here is to build a predictive model for the structure of forthcoming process instances. Technically, this model is conceived as a decision tree, which is constructed with an ad-hoc induction algorithm guaranteeing that the sooner an attribute tends to be known along the course of process enactments, the closer it appears to the root. Indeed, this feature is crucial to classify as soon as possible novel enactments at run-time and, hence, to use the decision tree as an actual forecasting tool. Finally, the synergical application of the two above techniques (for outlier detection and for building a predictive model, respectively) is illustrated with a simple example in Section 5, and over several real application scenarios in Section 6.

2. Related Work

Clustering in Process Mining Applications. Moving from the observation that classical process mining techniques often yield inaccurate "spaghetti-like" models when applied to a loosely-structured process, a recent research stream has proposed the clustering of log traces as a way to separately model different execution scenarios of the process itself [2–4]. In particular, in order to partition the log traces effectively, ad-hoc clustering approaches accounting for the peculiar nature of log traces have been devised, which actually record the execution over the time of possibly concurrent activities.

Most of the proposals in the literature focus on identifying an adequate similarity/dissimilarity measure for traces, in order to possibly reuse some existing distance-based clustering method [16]. For example, in [4] log traces are regarded as strings over the alphabet of task labels, and are compared via an edit distance measure quantifying their best (pairwise) mutual alignment. In particular, in order to go beyond a purely syntactical comparison of task labels and to make the clustering less sensitive to mismatches due to optional or parallel activities, the cost of each edit operation is chosen according to the context of the involved activities—where the context of a task is determined by the tri-grams (i.e., triples of consecutive activities) it appears in.

Instead of working directly on the sequential representation of log traces, some approaches [2, 3] map them in a feature space, where computationally efficient vector-based algorithm, such as k-means, can be used. For instance, different kinds of features (e.g., tasks, transitions, data attributes, performances) are considered in [3] for mapping each trace into a propositional representation, named "profile", possibly in a combined way. As specifically concerns structural aspects, two options are available for mapping a trace into a vector: (i) the task profile, where each task corresponds to a distinct feature and a sort of "bag-of-task" representation is obtained; and (ii) the transition profile, where the dimensions coincide with all possible pairs of consecutive tasks (i.e. a sort of bi-grams over the traces), viewed as a clue for inter-task dependencies. Looking at the usage of patterns for clustering complex objects and, in particular, sequential data (see, e.g., [17]), the latter approach may be well extended to accommodate more powerful structural features, such as sets of activities, higher order k-grams (with $k>2$), and generic (non-contiguous) subsequences. In fact, a special kind of pattern, based just on non-contiguous subsequences, is used in [2] to partition the log iteratively via the k-means algorithm.

Outlier-Detection. Outlier detection (a.k.a. anomaly detection, exception mining, deviation detection) is a major data analysis task, aimed at identifying instances of abnormal behavior [25]. Depending on the availability of labelled examples for normal and/or abnormal behavior, the outlier detection problem can be posed in three different settings: supervised, semi-supervised and unsupervised. Unsupervised approaches are the most closely related to the techniques discussed in this paper. Basically, they can be classified into three categories: model-based, NN-based, and clustering-based.

The first category, which embraces early statistics approaches to outlier detection (e.g., [19]), covers those approaches where some given kind of parametric or non-parametric distribution model is built that fits the given data, and where objects that do not conform enough with the model are pointed at as outliers.

NN-based methods (where NN stands for Nearest Neighbor) base instead the detection of outliers on comparing each object with its neighbors, according to either plain distances measures or to density measures. As an example of the first option, in [20], an object is classified as outlier if at least a fraction p of the data instances are more distant than r from it (for some user-defined thresholds p and r). Conversely, in [21] an outlier is an object whose density is quite lower than the average density of its neighbors (i.e., data instances falling within a given radius r)—where the density of an object is still computed based on its distance from its k-th nearest neighbor.

Finally, clustering-based approaches (e.g., [22–24]) assume that normal instances

form large (and/or dense) clusters, while anomalies belong either to very small clusters or to no cluster at all (or, equivalently, to singleton clusters). By the way, the adequateness of clustering algorithms for outlier detection is a somewhat controversial matter, specially as concerns algorithms, like k-means, which are rather sensitive to both noisy and anomalous instances, and which may fail to recognize adequately the real groups of normal objects in the dataset. To overcome this limitation, certain authors developed ad-hoc extensions of classical methods (e.g., the outlier-adaptive strategy of [23]), instead of simply using generic clustering algorithm more robust to noise and to outliers.

The above strategies have been extended to cope with complex data. We next only consider the cases of symbolic sequences and of process traces, due to their stronger connection with our work. Three main families of anomaly detection techniques have been proposed for symbolic sequences: Kernel-Based, Window-Based, and Markovian techniques. In the first case, an appropriate similarity measure (e.g., edit distance, longest common subsequence) is defined over the sequences, and existing distance-oriented anomaly detection techniques (e.g., NN-based or clustering-based) are trivially reused. In Window-Based techniques (e.g., [26]), a fixed size sliding window is used to logically split each input sequence s in smaller subsequences; the anomaly of s is then estimated by combining per-window (frequency-based) anomaly scores. Finally, Markovian approaches train a probabilistic generative model on the given sequences, assuming that the probability of any symbol in a sequence is conditioned on the symbols preceding it in the sequence. By estimating these per-symbol probabilities, one can compute the probability of the whole sequence, and derive an "outlierness" score for it (the lower the probability, the higher the score). Different kinds of model have been used in the literature to this end (e.g., k-order Markov chains [27], variable order Markov chains, usually kept as (probabilistic) suffix trees [28], and Hidden Markov Models [29]). Since all such approaches focus on the pure sequencing of tasks and assume that there exists some kind of dependence between elements appearing contiguously in a sequence, they will hardly work effectively in a process mining setting, where the traces in input may be generated by a non purely-sequential workflow process, where multiple parallel branches can proceed concurrently. See Section 3.1 and the last section in [40] for further remarks on this subject matter.

Primarily aimed at modelling normal behavior, classical process mining approaches gave little attention to outliers and to anomalies. In fact, most of these earlier approaches simply attempt to make the discovered control-flow model robust to the presence of noisy log records, by pruning unfrequent task dependencies, according to some user-driven threshold (see, e.g., [1, 13]). A few proposals appeared recently in the process mining community for the unsupervised detection of anomalies [30, 31], which remarked the importance of the task in flexible collaboration environments, particularly vulnerable to fraudulent/undesirable behaviors. The solution proposed in [30] consists in finding an appropriate workflow model, with the help of traditional control-flow mining techniques, and in judging a trace as anomalous if it does not conform with this model. Of course, the main limitation of this approach is that normal behavior is still modelled with the help of classical control-flow mining algorithms, which are not robust enough to the presence of several different outliers in the training log (but, at most, to records with random noise). Instead, in [31] the detection of outliers in a given log L relies on comparing any candidate (i.e. unfrequent) trace t with an

AND/OR graph-like model M_t, built in a dynamic way. Differently from [30], the model is induced from the subset $L - \{t\}$ (or from a random sample of it). The outlierness of t is then estimated either by verifying whether t is an instance of M_t, or by taking account for the structural changes required to make M_t represent t as well (the higher the cost, the more likely the trace is an outlier). A major drawback of this method is however that it requires to perform workflow induction for each candidate trace, thereby leading to prohibitive computational costs.

Supervised Classification via Decision Trees. Supervised classification aims at inducing a model for predicting which class (from a given set of a-priori classes) an object belongs to, based on other features of the object itself. This problem (a.k.a. *discriminant analysis* in classical statistics) has been targeted of intensive research for decades, giving rise to a great variety of alternative approaches (see, e.g., [32]).

Decision Trees are popular logic-based formalisms for representing classification models [16, 38]. A decision tree is a rooted directed tree, where each leaf is mapped to a class, while any internal node is associated with a test (decision) splitting the instances space based on the values of one or more attributes—in the latter case, the model is called *oblique* [33, 34]. Any (possibly novel) object can then be classified by following a suitable path from the root to one leaf, based on the outcome of the associated tests.

Inducing a decision tree from a given training set is an optimization problem where the goal is usually to minimize the generalization error, i.e., the error committed when classifying a test set. Other options are also possible such as to minimize the number of nodes or the average depth of the tree. However, finding a minimal decision tree is NP-hard (in any of these variants), and as such it claims for efficient search heuristics. In practice, decision trees are built by growing them iteratively according to *top-down* strategies. Essentially, a top-down induction algorithm initially builds a tree consisting of just one node, which contains the whole training set. Then, in a recursive way, a split test is chosen greedily for the current node, and applied to partition its associated instances. Different criteria have been proposed for guiding the splitting choice, which mainly rely on impurity measures (e.g., *Information Gain* [36], *Gini index* [33], and *Gain Ratio* [11]). The growth continues until either no more split is possible or some specific stop condition holds. This induction scheme is adopted, for example, by the classical algorithms *ID3* [12], *C4.5* [11], and *CART* [33] (where the selection of split tests is made according to Information Gain, Gain Ratio, and Gini index, respectively).

The choice of limiting the expansion of the tree is connected with the risk of *overfitting* [33] the training set, so that the model is unable to classify unseen records correctly. *Pruning* methods [33, 36] are an alternative solution. The basic idea is to first allow a complete growing of the tree; this possibly overfitted model is then trimmed by cutting portions of it that have low impact on the generalization error (i.e. the error made on unseen test instances). For example, an error-based pruning method is used in *C4.5* [11], whereas a cost-complexity mechanism is exploited by *CART* [33].

3. Outlier Detection in Process Mining Applications

3.1. Limitations of Existing Methods and Overview of the Approach

Outlier detection has already found important applications in bioinformatics fraud detection, and intrusion detection, just to cite a few. The basic observation underlying

the various approaches is that abnormality of outliers cannot, in general, be defined in "absolute" terms, since outliers show up as 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 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 (e.g. [27–29]) to process logs may suffer from a rather high rate of *false positives*, as a notable fraction of task sequences might have very low frequency in the log. For example, in Figure 1, each of the sequences $s_1, ..., s_5$ rarely occurs in the log, but should not be classified as anomalous, as they are different interleaving of a single enactment, which occurs in 10 out of 43 traces. As an extreme case, consider an additional trace t_{new} exhibiting the task sequence `acdflegh`, which conceptually corresponds to the same execution scenario as the sequences $s_6, \ldots, s_9$. When a Markov chain model is learned from the example log, like in [27, 28], the probability score estimated for t_{new} will be very low, since no other trace in the training log features the subsequence *le*, and this trace will be incorrectly deemed as an outlier. Further details on this matter can be found in [40].

(C2) On the other hand, considering the compliance with an ideal (possibly concurrent) worklfow schema might lead to *false negatives*, since some trace might well be supported by a model, even though it identifies a behavior that deviates from the one observed in the majority of the traces. For example, in Figure 1, trace s_{16} corresponds to cases where all the tasks but `b` were executed. Even though this behavior is admitted by the model in the same figure, it is anomalous since it only occurs in 2 out of 43 traces. In addition, when such an ideal schema is not known a-priori and classical workflow discovery techniques are used for its discovery (as proposed in [30]), the presence of several outliers in the training instances may well lead to a distorted model of normal behaviors, which will eventually produce both *false negatives* and *false positives*.

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. Indeed, the key 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 accurately describing 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*. Outliers are then identified in a two-steps approach:

- first, *patterns* of executions are mined which 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 effectively characterizing concurrent processes, and by *(ii)* presenting an algorithm for their identification;

- second, an outlier detection approach is used which is *cluster-based*, i.e., it computes a clustering for the logs 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.

Note that a key point in the methodology concerns the kinds of patterns adopted for the clustering task. In fact, the usage of basic structural elements extracted from the traces (such as activities or pairs of contiguous activities like in [3], or the sequential patterns introduced in [2]) completely disregards the concurrent semantics of process logs—where parallel execution branches may be registered in an interleaved way—and risks not to recognize adequately the groups of traces corresponding to different execution scenarios. The above two-steps methodology, instead, reduces 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)**. Moreover, in order to better deal with high-dimensionality and with the uncertainty linked to both noise and parallelism, patterns are not used to map the traces into a vectorial space (as in [2, 3]) where classic clustering methods can be applied, but rather a sort of coclustering method is adopted which focuses on the association between traces and patterns.

The above techniques are illustrated in Section 3.2, while some basic algorithmic issues are discussed in the subsequent Section 3.3.

3.2. Formal Framework for Outlier Detection

Process-aware commercial systems usually store information on process enactments by tracing the events related to the execution of the various tasks. Abstracting from the specificity of the various systems, as commonly done in the literature, we view a *log* L over a set of tasks T as a bag of *traces* over T, where each trace t in L is a sequence of the form $t[1]t[2]...t[n]$, with $t[i] \in T$ for each $1 \leq i \leq n$. Next, a log is assumed to be given and the problem of identifying anomalies in it 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., [8]). 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 behavior, 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 tasks. This is, for instance, the case of patterns p_3, p_4, and p_6 in Figure 1. Instead, higher-sized patterns account for fork and join constructs, specifying parallel execution (cf. p_2) and synchronization (cf. p_5), respectively, in 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 t includes all the tasks in T_p, and the projection of t over T_p is a topological sorting of p, i.e., there are not 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 other 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, according to a negative exponential law, 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. Essentially, the usage of a negative exponential weighting function allows for a more aggressive penalization of the support score when an increasing number of such spurious tasks appears in the log traces. In fact, the superiority of such a choice to other (e.g., linear) weighting schemes was confirmed by some tests on synthesized logs.

Example 1. Consider again the example in Figure 1. It is clear that all traces corresponding to any of the sequences $s_{10}, \ldots, s_{15}$ do not comply with p_1. For the remaining traces, the application of the support function defined above gives the following results:

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

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

While at a first sight the above 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 increasing its support,

since one further task (either *x* or *y*) may be no longer viewed as a spurious one. Consequently, in order to find all patterns with support greater than a given threshold σ (hereinafter called σ-*frequent* patterns), we cannot simply reuse classical *level-wise* approaches (like the popular Apriori algorithm), which efficiently prune portions of the lattice[3] of all the possible *S*-patterns— by exploiting the *anti-monotonicity* property (a.k.a. downward closure) of classical frequency-oriented support measures (i.e., for any frequent pattern all of its sub-pattern must be frequent as well). 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 modelling process behavior. Instead, the relevance of a pattern is captured in the following definition.

Definition 3 (Interesting Patterns). Let L be a log, σ, γ be two real numbers. Given two *S*-patterns p and p', we say that p' γ-*subsumes* p_1, denoted by $p \sqsubseteq_\gamma p'$, if p is a subgraph of p' and $supp(p,L) - supp(p',L) \leq \gamma \times supp(p',L)$. 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 $p \sqsubseteq_\gamma p'$. □

In other words, we are not interested in a frequent pattern p if it its frequency is not significantly different from that of some other pattern p' that includes it; conversely, if p is much more frequent than p', one can assume that the subpattern p has its own value in characterizing the behavior of the process. Notice that, when testing the maximality of pattern p, this is only compared with patterns having just one more edge, in order to curb computational costs. However, owing to the peculiarity of the support function adopted here, there might be some bigger pattern (having multiple edges more than p) that violates condition (b). Anyway, the above notion of maximality is expected to suffice to filter out a large portion of uninteresting patterns.

Example 2. Let us consider the patterns p_5 and p_1 in Figure 1, 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 that of p_2. ◁

Clusters-Based Outliers. Once that "normality" has been roughly captured by means of frequent patterns, one can look for those individuals whose behavior deviates from the normal one. To this end, the second step of the outlier detection approach is based on a *coclustering* (see, e.g., [9]) method for simultaneously clustering both patterns and traces, on the basis of their mutual correlation, as it is expressed by the measure *supp*. Intuitively, pattern clusters are to be associated 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 (see Section 3.3), the output of this method is formalized below.

[3] A lattice is a partially ordered set of elements, whose union and intersection are the least upper bound and the greatest lower bound, respectively, for all elements.

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

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

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

Outliers have been defined above 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 1. 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, s_8, s_9 do the same with both patterns p_1 and p_3. Moreover, s_{10} highly supports both p_3 and p_6, whereas s_{11} is strongly associated with both p_4 and p_6. Finally, sequence s_{16} is associated with all of the patterns in Figure 1 but p_4 and p_6. By using some suitable coclustering 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}, which will likely originate a separate cluster just consisting of the two traces that correspond to s_{16}. Yet, 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. This can be accomplished by setting the threshold β, controlling the minimal cluster size, in a way that this small cluster is regarded as a set of outliers (as in many clustering-based outlier detection approaches). ◁

3.3. `OASC`: an Algorithm for Detecting Outliers in a Process Log

In this section, we discuss an algorithm, named `OASC`, for singling out a set of outliers, based on the computation scheme and the framework described so far.

The algorithm, shown in Figure 2, takes in input a log L, a natural number *pattSize* and four real thresholds σ,γ, α and β. The algorithm first uses 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

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

Function `FindPatterns`(L: log; $pattSize$: natural number; σ, γ : real numbers): 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** $M(i,j) := \frac{|\{t' \in L \mid supp(p_i,t') \geq \alpha \wedge supp(p_j,t') \geq \alpha\}|}{|\{t' \in L \mid supp(p_i,t') \geq \alpha \vee supp(p_j,t') \geq \alpha\}|}$
C2 Compute a partition P_{mcl} of P by applying the *MCL* clustering algorithm to M;
C3 $L := \emptyset$; $P := \emptyset$; $M := \emptyset$;
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 thus associated with some trace cluster*
C7 Let $l^t = M(p^t)$ be the cluster currently associated with p^t, and $l^t_{new} = l^t \cup \{t\}$;
C8 $L := L - \{l^t\} \cup \{l^t_{new}\}$; $M(p^t) := l^t_{new}$;
C9 **else**
C10 $L := L \cup \{\{t\}\}$; $P := P \cup \{p^t\}$; $M(p^t) := \{t\}$;
C11 **end if**
C12 **end for**
C13 **return** $\langle P, L, M \rangle$;

Figure 2: Algorithm `OASC`.

is extracted with the function `FindCoClusters` (Step 2). The subsequent 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.

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 $supp'$) 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 higher 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.

Notice 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 explained next. 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 compute incrementally 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 applying 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, `FindCoClusters` computes an α-coclustering $\langle P, L, M \rangle$ for L and P, where P (resp., L) is a set of pattern (resp., trace) clusters, while M is a mapping from P to L.

At the 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* [10], which has been proved to achieves good results on several large datasets and selects autonomously the number of clusters.

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 P, for it was induced by some other traces; in this case we retrieve, by using the mapping 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 L, and update M to store the association between this new cluster and p^t, which is stored as well in P as a novel pattern cluster (Step C10).

Complexity issues. Assume that a log of N traces over T tasks is given as input. Let P_{can} be the maximum number of patterns found at any iteration of the loop in function `FindPatterns`, and let P_{max} be the number of patterns used for the clustering in `FindCoClusters`. Then, the complexity of `OASC` is essentially given by the expression

$O(N \times (T + P_{can} \times S^2) + T \times P_{can} + P_{max} \times K^2)$, where S is the maximal size of patterns (i.e., $pattSize = S$) and K is a parameter of algorithm *MCL* [10] (cf. Step C2)—a detailed complexity analysis is reported in [40]. Note that P_{can} might in principle be exponential in T. However, in real cases where process tasks obey precise routing rules and unfrequent patterns are pruned via the support threshold σ, the number of candidate patterns is unlikely to blow up. Moreover, notice that `OASC` requires a limited number of scans over the input log, and does not need to keep it into the main memory. Indeed, the log can just be scanned S times ($S < 10$ worked fine in our experiments) to find patterns of size S, plus two further times to build matrix M and assign the traces to clusters (Steps C4-C12). This property guarantees potential scaling over huge datasets.

4. Discovery of Context-based Predictive Models

After a set L_* of trace clusters has been computed, the natural question comes into play about whether one can find a model predicting the membership into the various clusters based on the (non-structural) data available for the process instances at hand. By conceiving the predictive model as a decision tree and by regarding the clusters as different classes of traces, this problem amounts at inducing a decision tree from the given log, provided that each trace in the log has been marked with the label of the cluster it was assigned to (in the clustering phase). In particular, while inducing the decision tree, it is desirable that the decisions in the model are primarily based on attributes that are likely to be known in the earlier steps of a process enactment, in order to possibly make prediction even on uncompleted process instances. This feature is very peculiar to process mining applications, and calls for developing ad-hoc induction algorithms that are aware of the precedence relations over the activities (as it is inferred from the log). This issue, which has been not addressed in the earlier literature (see Section 2), will be faced in the rest of the section.

4.1. Formal Framework for the Induction of Predictive Models

In principle, process logs may contain a wide range of information on process executions. The notion of log traces used so far is then extended next 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 τ (i.e., $\tau = task(a)$), we will sometimes refer to a as $\tau.a$, in order to represent its association with τ compactly and intuitively. 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 τ being executed, the set of all its activities will be mapped to some values taken from the respective domains.

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 $t \in L$ to 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 $\{a_1, ..., a_q\} = \{a \in A \mid task(a) = t[j], \text{for some task } t[j] \in T\}$. □

Next, we assume that the set 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 (*DADT* Model). Let L_* be a set of trace clusters (for a data-aware log) over a set T of tasks and a set A of associated attributes. Then, a *data-aware decision tree* (shortly, *DADT*) for L_* is a triple $D = \langle H, \texttt{attr}, \texttt{split}, \texttt{pred} \rangle$ where:

- $H = (N, E)$ 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 $\texttt{attr}(v)$;
- $\texttt{pred} : N \times L_* \to R$ is a function expressing the probability of any cluster in L_* conditioned on each node in N[4]. □

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* $D = \langle H, \texttt{attr}, \texttt{split}, \texttt{pred} \rangle$, 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 $l \in L_*$ 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 L_* to the clusters that are σ'-active in v is denoted by $L_*(\sigma', v)$. Moreover, given two tasks τ and τ', we say that τ σ'-*precedes* τ' in l, denoted by $\tau \prec^{l}_{\sigma'} \tau'$, if at least a fraction σ' of l's traces contain τ before τ', and less than a fraction σ' of l's traces contain τ after τ'.

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

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 $L_*(\sigma', v)$). 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 constraints guarantee that σ'-compliant *DADT*s are suitable models to support on-the-fly prediction.

[4] Thus, function $\texttt{pred}$ can be used to predict the structural cluster associated with each node v of the tree, as the one having the maximal conditional probability w.r.t. v.

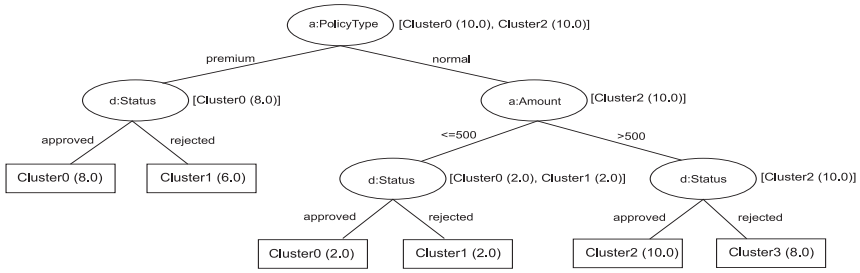


Figure 3: A 0-compliant *DADT* found by algorithm `LearnDADT` ($\omega = 0.35$).

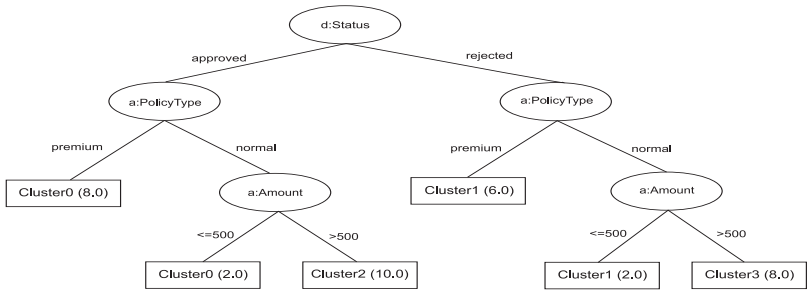


Figure 4: A decision tree found without considering temporal aspects (i.e., $\omega = 1$).

Example 4. Two *DADT* models for the log in Figure 1 are shown in Figures 3 and 4, where the mapping from nodes to data attributes and to predicted clusters (i.e. functions `attr` and `pred`, respectively) and the association of edges with split formulae (i.e., function `split`) are all indicated informally via intuitive node/edge annotations. Assuming $\sigma' = 0$ for simplicity, it is easy to see that the model in Figure 3 is σ'-compliant, whereas the other is not. This latter fact can be verified by noting that conditions (a) and (b) in Definition 7 do not hold on the root of the tree and its left child. $\lhd$

4.2. `LearnDADT`: an Algorithm for Inducing a DADT Model

Several decision-tree induction approaches are already available in the literature that might be used, in principle, to built a compliant *DADT*. However, However, by straightforwardly integrating the compliance constraint into them, one risks 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. In this case, Definition 7 would allow further partitioning of the training set based only on attributes of *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, and we introduce a bias towards attributes of tasks that were executed in earlier phases of past process enactments. This is mainly

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

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

Procedure `growDT`(D: a *DADT*, v: a D's node, S: a set of traces; 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 π* be the split formula evaluated for a*;
B5     if ω < 1 and checkCompliance(a*,D,v,L'_σ)
B6       D.split(v) := π*; D.attr(v) := a*;
B7       let S_1,...,S_k be the partition of S obtained by applying the test π* 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
B12  for each l_i ∈ L_* do D.pred(v,l_i) := |l_i ∩ S|/|S|;
```

Figure 5: Algorithm `LearnDADT`.

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 above strategy is shown in Figure 5. The algorithm starts building a preliminary *DADT* that just consists of one node (named *r* in the figure), gathering all log traces (the set *L* contains indeed the traces of all clusters in input). Then (line 2) a decision tree is built in a top-down way, via a recursive partitioning procedure, named `growDT`, which will be discussed in detail later. Once such a (possibly large and overfitted) decision tree is built, a pruning procedure (similar to the "subtree replacement" method of algorithm J48 [39]), is exploited to ensure accurate prediction even on new process instances. The pruned DADT model is returned as the ultimate outcome of the algorithm.

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 D, the leaf node *v* and its associated set *S* of traces, which are to be considered for splitting, 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*, this split score is computed as a linear combination (with weight $\omega \in [0..1]$) of two components:

- A predictiveness measure, denoted by $g(a,S)$ and computed through the classical Gain Ratio measure [12], mainly accounting for the reduction of information entropy that descends from splitting *S* based on the values of *a*.
- An ad-hoc score ep that takes account for the dynamical aspects of the process,

by introducing a bias towards attributes that are associated with tasks that occur earlier in the traces corresponding to clusters correlated with v significantly.

More precisely, denoting by L'_σ the set of 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:

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

where $tasks(l)$ stands for the set of tasks appearing in the traces of cluster l, while $succ(task(a),l)$ is the set of tasks in $task(l)$ that follow $task(a)$ under the ordering relationship $\prec^l_{\sigma'}$, i.e., $succ(task(a),l) = \{t' \in tasks(l) \mid t \prec^l_{\sigma'} t'\}$.

We pinpoint that when making *score* coincides 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^*, D, v, and L'_σ will return `false` iff *(i)* there is an ancestor v' of v in D such that $task(v')$ precedes a^* in some cluster of C'_σ, and *(ii)* there is no ancestor v'' of v' in 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 L given as input to the algorithm. To this aim, one could think of resorting to some kind of workflow model (possibly discovered through classical process mining techniques, such as those presented in [1]). Since the compliance test is done only when $\omega < 1$, the behavior of algorithm `LearnDADT` is made to coincide with that of traditional decision tree learning algorithms in the case where $\omega = 1$.

In the case the check performed by `checkCompliance` is passed successfully, the current (leaf) node v is mapped to both the selected split formula π^* and the associated attribute a^*, by suitably updating the functions *split* and `attr` of the *DADT* D (line B6). 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 π^* to S (lines B7-B8).

Then, the procedure `growDT` is recursively applied to each new node v_i, and its corresponding set of traces S_i. Finally, the probability of any cluster l_i conditioned on node v is estimated as the relative frequency of l_i in S (line B12).

Complexity issues. The computation cost of algorithm `LearnDADT` is $O(N \times (H \times F + T^2))$, where H is the height of the tree created by procedure `growDT`, and N, T and F are the numbers of traces, tasks and attributes, respectively, in the input log—further details can be found in [40]. As in standard decision tree induction approaches (see, e.g., [39]), the tree grown is rarely complete and $H \ll \log(N)$ holds. Thus, `LearnDADT` often takes linear time in N. Nonetheless, in order to deal efficiently with large logs, we are investigating the usage of external-memory (possibly parallel) DT-induction approaches (see [38] for detailed references), some of which (e.g. *RainForest*) can naturally combine with our C4.5-like scheme.

5. Putting It All Together: A Toy Application Example

Consider the log in Figure 6, which is a refined representation of the one in Figure 1 where each trace is associated with non-structural data encoded as attribute-value pairs. The log concerns the processing of liability claims in an insurance company [6]. The behavior of the underlying process is 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 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 to eventually issue a payment for the claim: **f** (Submit Payment), **l** (Validate Payment), **m** (Update Reserves), **n** (Send Notification), **g** (Register Payment). Finally, the claim is archived and closed (task **h**, Archive claim).

trace ID	task sequence	data
t_1	s_1 : abdfnmlgeh	{(a.Amount,1000),(a.PolicyType,premium),(d.Status,approved)}
t_2	s_1 : abdfnmlgeh	{(a.Amount,1050),(a.PolicyType,premium),(d.Status,approved)}
t_3	s_2 : abdfelmngh	{(a.Amount,5000),(a.PolicyType,premium),(d.Status,approved)}
t_4	s_2 : abdfelmngh	{(a.Amount,500),(a.PolicyType,premium),(d.Status,approved)}
t_5	s_3 : abdeflmngh	{(a.Amount,495),(a.PolicyType,premium),(d.Status,approved)}
t_6	s_3 : abdeflmngh	{(a.Amount,500),(a.PolicyType,normal),(d.Status,approved)}
t_7	s_3 : abdeflmngh	{(a.Amount,480),(a.PolicyType,normal),(d.Status,approved)}
t_8	s_4 : abdfmnlgeh	{(a.Amount,6000),(a.PolicyType,premium),(d.Status,approved)}
t_9	s_5 : abdeflnmgh	{(a.Amount,6200),(a.PolicyType,premium),(d.Status,approved)}
t_{10}	s_5 : abdeflnmgh	{(a.Amount,5800),(a.PolicyType,premium),(d.Status,approved)}
t_{11}	s_6 : acdfmlgeh	{(a.Amount,500),(a.PolicyType,normal),(d.Status,rejected)}
t_{12}	s_6 : acdfmlgeh	{(a.Amount,490),(a.PolicyType,normal),(d.Status,rejected)}
t_{13}	s_7 : acdfelmgh	{(a.Amount,600),(a.PolicyType,premium),(d.Status,rejected)}
t_{14}	s_7 : acdfelmgh	{(a.Amount,610),(a.PolicyType,premium),(d.Status,rejected)}
t_{15}	s_8 : acdeflmgh	{(a.Amount,615),(a.PolicyType,premium),(d.Status,rejected)}
t_{16}	s_8 : acdeflmgh	{(a.Amount,605),(a.PolicyType,premium),(d.Status,rejected)}
t_{17}	s_8 : acdeflmgh	{(a.Amount,620),(a.PolicyType,premium),(d.Status,rejected)}
t_{18}	s_9 : acdfmlgeh	{(a.Amount,400),(a.PolicyType,premium),(d.Status,rejected)}
t_{19}	s_{10} : acdih	{(a.Amount,501),(a.PolicyType,normal),(d.Status,approved)}
t_{20}	s_{10} : acdih	{(a.Amount,555),(a.PolicyType,normal),(d.Status,approved)}
t_{21}	s_{10} : acdih	{(a.Amount,560),(a.PolicyType,normal),(d.Status,approved)}
t_{22}	s_{10} : acdih	{(a.Amount,565),(a.PolicyType,normal),(d.Status,approved)}
t_{23}	s_{10} : acdih	{(a.Amount,570),(a.PolicyType,normal),(d.Status,approved)}
t_{24}	s_{10} : acdih	{(a.Amount,575),(a.PolicyType,normal),(d.Status,approved)}
t_{25}	s_{10} : acdih	{(a.Amount,580),(a.PolicyType,normal),(d.Status,approved)}
t_{26}	s_{10} : acdih	{(a.Amount,585),(a.PolicyType,normal),(d.Status,approved)}
t_{27}	s_{10} : acdih	{(a.Amount,590),(a.PolicyType,normal),(d.Status,approved)}
t_{28}	s_{10} : acdih	{(a.Amount,595),(a.PolicyType,normal),(d.Status,approved)}
t_{29}	s_{11} : abdih	{(a.Amount,550),(a.PolicyType,normal),(d.Status,rejected)}
t_{30}	s_{11} : abdih	{(a.Amount,545),(a.PolicyType,normal),(d.Status,rejected)}
t_{31}	s_{11} : abdih	{(a.Amount,540),(a.PolicyType,normal),(d.Status,rejected)}
t_{32}	s_{11} : abdih	{(a.Amount,535),(a.PolicyType,normal),(d.Status,rejected)}
t_{33}	s_{11} : abdih	{(a.Amount,530),(a.PolicyType,normal),(d.Status,rejected)}
t_{34}	s_{11} : abdih	{(a.Amount,525),(a.PolicyType,normal),(d.Status,rejected)}
t_{35}	s_{11} : abdih	{(a.Amount,520),(a.PolicyType,normal),(d.Status,rejected)}
t_{36}	s_{11} : abdih	{(a.Amount,501),(a.PolicyType,normal),(d.Status,rejected)}
t_{37}	s_{12} : afih	{(a.Amount,641),(a.PolicyType,normal)}
t_{38}	s_{13} : ah	{(a.Amount,520),(a.PolicyType,normal)}
t_{39}	s_{14} : aeg	{(a.Amount,580),(a.PolicyType,normal)}
t_{40}	s_{14} : aeg	{(a.Amount,700),(a.PolicyType,normal)}
t_{41}	s_{15} : adfemh	{(a.Amount,1000),(a.PolicyType,normal),(d.Status,rejected)}
t_{42}	s_{16} : acdfmenlgh	{(a.Amount,0),(a.PolicyType,normal),(d.Status,rejected)}
t_{43}	s_{16} : acdfmenlgh	{(a.Amount,0),(a.PolicyType,normal),(d.Status,rejected)}

Figure 6: Example log for a claim handling process.

Notice that only the activities **a** and **d** have data items associated with them: 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**). In particular, *Amount* is a numerical attribute, while *PolicyType* and *Status* are nominals taking values from {"*normal*","*premium*"}, and {"*approved*","*rejected*"}, respectively.

Discovery of Behavioral Clusters and Outliers. Let us first examine the behavior of algorithm OASC against the example log of Figure 1, with $\sigma = 0.1$, $\gamma = 0.2$, $\alpha = 0.4$, and $\beta = 0.3$. The algorithm discovers 4 different structural clusters: one with the traces $t_1, ...t_{10}$ (corresponding to the sequences $s_1, ..., s_5$ of Figure 1), 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_{10}), and one with the traces $t_{29}, ..., t_{36}$ (corresponding to s_{11}). The remaining log traces are recognized as anomalous process instances, which is in line with the observations made in Example 3 concerning desirable outcomes of such a clustering process. For instance, although s_{16} trivially induces a cluster with two traces, these are perceived as outliers since the cluster is too small (w.r.t. to the average cluster size and β). Note that we have also experimented the application of other clustering approaches [3, 4] to the same example log, taking advantage of their respective implementations provided in ProM [14]—actually, we used several clustering procedures (including k-means/medoid, and agglomerative clustering schemes with different linkage options) for both [4] and [3], and several similarity/distance measures for the latter. However, in none of these trials we were able to find the same partition of the log as the one found by algorithm OASC (whatever was the number of clusters asked as input, the cut applied to the resulting dendrogram).

The four clusters obtained via algorithm OASC have been subsequently processed by using the *HeuristicMiner* plugin [13] available in the process mining framework ProM [14], in order to associate a workflow with each of them modelling the behavior of the associated traces. Figure 7 shows the resulting workflow models[5], actually representing four major execution scenarios for the process itself, which mainly differ in the kind of policy check performed — task **b** (Check policy only) vs. task **c** (Check all) — and in the final decision on the claim — task **e** (Send Approval Letter) vs. task **i** (Send Rejection Letter). Discovering these usage scenarios improves 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[5] shown in Figure 8, which was obtained by directly applying the *HeuristicMiner* plug-in to the whole log of Figure 6. Beside modelling some additional spurious task links (due to the presence of outlier traces $t_{37}, ..., t_{41}$), this workflow schema incorrectly allows for simultaneously executing the tasks **e** (Send accept letter) and **i** (Send rejection letter), although they occur together only 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 of the claim was accomplished, by way of task **c** (Check all)[6].

[5] By each workflow node the respective task ID is reported in a magnified form, for readability reasons.

[6] It is worth noting that these behavioral rules, effectively captured via our clustering-oriented approach,

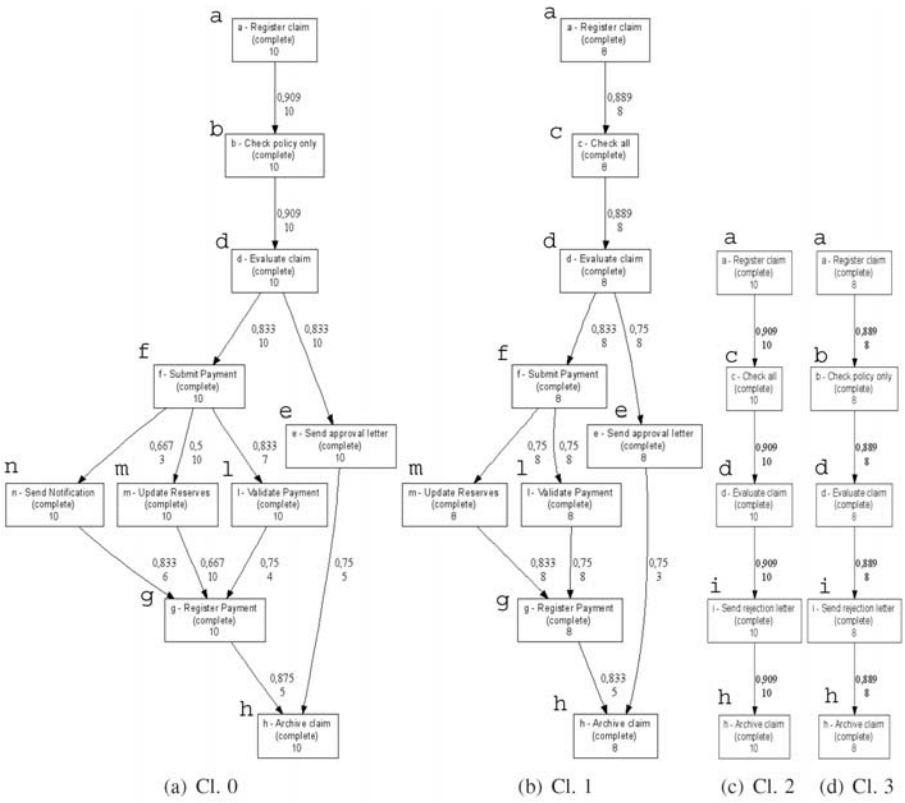


Figure 7: Usage scenarios for the example in Section 5.

Discovery of Predictive Models. Let us now apply algorithm `LearnDADT` to the clusters found by algorithm `OASC`, as to find a predictive model expressing the correlation of these behavioral classes with non structural process attributes. To this end, we retained all data attributes but *CustomerID* (which is indeed useless for learning general behavior). Moreover, we fixed $minCard = 0$ and $\sigma' = 0.05$, while considering two different values for ω, namely $\omega = 0.35$ and $\omega = 1$. The models returned in the two cases are sketched in Figure 3 and Figure 4, respectively. Note that $\omega = 1$ practically corresponds to applying the classical decision-tree induction algorithm C4.5 [11]. In fact, differently from the tree that is inferred in this basic case, the topology of the model in Figure 3 fits well the task precedences expressed by the schemas of Figure 7. Incidentally, this result has been achieved without incurring any loss in the accuracy of the model (w.r.t. the input log)—which is maximal for both trees in Figures 3 and 4.

Note that any node of the tree in Figure 3 is associated with a probability function relating the node itself with each cluster. Actually, beside each non-leaf node *v*, we only report the most probable clusters for *v* and the number of log traces

correspond to very complicated workflow patterns (involving non-local task dependencies and hidden tasks) that are beyond the scope of most process mining approaches [13].

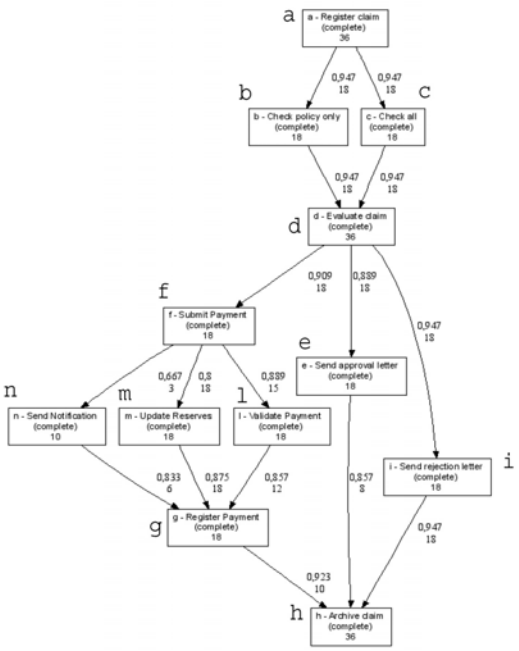


Figure 8: A schema mixing the various scenarios on the example in Section 5.

that felt in *v* during the learning process. Clearly enough, such information can be exploited to predict cluster membership for ongoing process instances. For example, one can exploit the tree in Figure 3 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*. In fact, in addition to its predictive capabilities, a model like the one in Figure 3 has an evident descriptive value, and can help to interpret the execution scenarios discovered for the given process in term of non-structural aspects of the process itself.

6. Experiments

The approach proposed in the paper has been implemented and integrated into a Java prototype system, which is meant to support the analysis of process logs represented in the MXML format used in the ProM process mining framework [14]. In particular, the system can be exploited to detect structurally homogeneous trace classes and anomalous traces in a given input log, as well as to discover a decision tree model for predicting class membership based on context data. This section discusses the application of the system on two different real-life scenarios, with the aim of providing evidence for the practical usefulness of our proposal.

The rest of this section is organized as follows; in Section 6.1, a series of metrics enabling for a quantitative evaluation of experimental findings are introduced; concrete datasets used in the experimentation are illustrated in Section 6.2. Section 6.3 offers a summarized view over the experiments, while further results are discussed in [40].

the capability of the method to support "on-the-fly" prediction over uncompleted process instance is evaluated in Section 6.4, while Section 6.5 finally illustrates results of experimental activities meant to analyze the effectiveness of the approach in detecting a-priori known clusters and outliers, and to study the impact of input parameters.

6.1. Evaluation Setting

In the evaluation of experimental results we focused on the *(i)* 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 on the *(ii)* 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. The metrics adopted to this end are illustrated next.

Quality of Structural Models. The conformance of a workflow model W w.r.t. a log L can be measured via two complementary metrics (defined in [15]), ranging over the real interval [0..1]: the *fitness* (denoted by *Ft*), which essentially evaluates the ability of W to parse all the traces L, by indicating how much the events in L comply with W; and the *advanced behavioral appropriateness* (denoted by *BA*), which estimates the level of flexibility of W (i.e., alternative/parallel behavior) used to produce L. These measures are defined for workflow schemas and do not apply directly to log clusters. Thus, each cluster identified via `OASC` is equipped with a workflow schema, by using the *HeuristicMiner* plugin [13] provided by the process mining framework ProM [14]—this choice, which is actually orthogonal to our approach, mainly descends from the fact that this plugin is robust to noise and efficient enough. Then, *Ft* (resp., *BA*) is computed by summing up the fitness (resp., advanced behavioral appropriateness) of each induced workflow schema, where the weight of each schema is the fraction of the original log traces constituting the cluster it was mined from.

Prediction Quality. To evaluate the precision of *DADT* models, we use 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 [16]). This measure is also computed on incomplete log traces, in order to assess the capability of *DADT* models to carry out "on-the-fly" predictions (see Section 6.4). Moreover, as a further measure, a score is introduced indicating how much the model complies with precedence relationships among tasks. In order to make the evaluation independent of discovered models, we only compute it against the log, by measuring, for each leaf node l and for each trace t assigned to l, how 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 (DADT Conformance). Let L be a log over task set T and attribute set A, and $D = \langle H, \texttt{attr}, \texttt{split}, \texttt{pred} \rangle$ be a DADT model. For any leaf ν of $D.H$, let *(i)* $a_1^\nu \ldots a_k^\nu$ be the attributes associated with all non-leaf nodes $n_1^\nu \ldots n_k^\nu$ in the path from $D.H$'s root to ν—i.e., $a_i^\nu = D.\texttt{attr}(n_i^\nu)$, for $i = 1..k$—, and *(ii)* $path(\nu) = p_1^\nu, \ldots, p_k^\nu$ be the sequence of tasks corresponding to $a_1^\nu \ldots a_k^\nu$—i.e., $p_i^\nu = task(a_i^\nu)$ for $i = 1..k$. Then, the *conformance of* D *w.r.t.* L, denoted by $Conf(D,L)$ is defined as follows:

$$Conf(D,L) = \frac{1}{N} \sum_{\nu \in leaves(D.H)} \sum_{t \in traces(\nu)} \left(1 - \frac{mismatches(t, path(\nu))}{maxMismatches(t, path(\nu))}\right)$$

where $leaves(H)$ and $traces(\nu)$ simply denote the leaf nodes of the tree H and the log traces assigned to its leaf node ν, respectively; $mismatches(t, path(\nu))$ is the number of times the task precedences in t are inverted in $path(\nu)$, and $maxMismatches(t, path(\nu))$ is the maximum number of such inversions that may occur between two sequences containing the same tasks as t and $path(\nu)$, respectively[7]. Moreover, for any *DADT* model $D = \langle D, \texttt{attr}, split, p \rangle$, we will also denote $Conf(D, L) = Conf(D.D, L)$. □

Essentially, this score is meant to evaluate how much a DADT model agrees with the actual ordering of tasks in the log traces, and gives a rough estimate of its ability to make accurate predictions over an ongoing process instance.

Example 5. Consider the decision trees in Figures 3 and 4, and the example log in Figure 6. Let ν_1^a and ν_1^b indicate the leftmost leaf in the tree of Figure 4 and of Figure 3, respectively. Let us also denote by t_1 the first trace in the log of Figure 6, which clearly corresponds to the task sequence `abdflenmgh`. Note that t_1 is assigned to ν_1^a (resp., ν_1^b) in the tree in Figure 3 (resp., 4), which corresponds to the task sequence $path(\nu_1^a) =$ `ad` (resp., $path(\nu_1^b) =$ `da`). Therefore, it holds that $mismatches(t_1, path(\nu_1^a)) = 1$ and $mismatches(t_1, path(\nu_1^b)) = 0$. In fact, the same happens for the whole log, given that in all paths in Figure 4 leading to leaves task `d` precedes task `a`, while this ordering is violated in all the traces. Therefore, the overall conformance measure *Conf* is 0 for the tree in Figure 4, and 1 for that in Figure 3. ◁

Note that the measure $Conf(D, L)$ defined above is a pessimistic estimate for the capability of a *DADT* D to comply with the workflow models that could be discovered for the log L, by using some suitable process mining technique. For instance, if 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* D that uses both tasks, the $Conf(D, L)$ will incorrectly count a mismatch on either t_{db} or t_{bd}.

6.2. Datasets

Experimental activities were carried out on datasets from two different real-life application scenarios, which are described next.

Data From a Logistic System (Logs A and B). The first application scenario concerns the operational system used in an Italian maritime container terminal. The life cycle of any container is 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 train-like vehicle that can transport multiple containers). Each container undergoes several logistic operations determining its displacement across the "yard"—i.e., the main area used in the

[7] $maxMismatches(t, path(\nu)) = min\{|t|, |path(\nu)|\} \times (min\{|t|, |path(\nu)|\} - 1)/2$, where $|t|$ (resp., $|path(\nu)|$) denotes the number of distinct tasks appearing in t (resp., $path(\nu)$).

harbor for storage purposes, logically partitioned into bi-dimensional *slots*. Slots are 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 to 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: *(i)* ***Log A*** ("operation-centric"), storing the sequence of logistic operations applied to the containers; and *(ii)* ***Log B*** ("position-centric"), registering the flow of containers across the yard. In addition, various data attributes were considered for each container, 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). These data were encoded as attributes of the starting task.

Data From a Collaboration Process (Log CAD). The second application scenario, studied in the research project *TOCAI.it*[8], concerns the collaborative process performed in a manufacturing enterprise in order to carry out the design, 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. The following kinds 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), `Prototyping` (a prototype was built for an item), `Test` (the project was validated), `TechRevision` (a technical revision was done for an item), `Share` (the project was shared with other workers), `Release` (the project was released), `PilotSeries` (a pilot series was produced).

In particular, we focused on the operations performed in the first three months of year 2007, over 5794 projects. These historical data 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 with two attributes, concerning the user that performed it: the work group he/she belonged to (*Group*), and the role he/she was playing in the design process (*Role*).

6.3. Experimental Results: Quality of Discovered Models

Table 1, Table 2 and Table 3 summarize the outcomes of a selection of experiments performed on the log data described above. In particular, Table 1 reports the number of clusters found by `OASC`, and the quality scores associated with them for the threshold values σ=σ'=0.05, γ=4, α=0.4, and β=0.1. In these tests, the recognition of various kinds of outliers allowed to achieve high quality workflow models for the behavioral

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

Log	Clusters	Outliers	Ft	BA
Log A	2	53	0.8725	0.9024
Log B	5	63	0.8558	0.9140
Log CAD	4	50	0.6842	0.6584

Table 1: Results of algorithm OASC.

Method	*Log A*			*Log B*			*Log CAD*		
	#Cl	Ft	BA	#Cl	Ft	BA	#Cl	Ft	BA
OASC	2	0.8725	0.9024	5	0.8558	0.9140	4	0.6842	0.6584
OASC—no outliers	2	0.8320	0.8821	10	0.7947	0.8247	4	0.6421	0.6341
Feature-based [3]	2	0.8716	0.8842	5	0.7332	0.8121	4	0.6031	0.6341
Edit-based [4]	1	0.8301	0.7631	5	0.8423	0.9076	4	0.6828	0.6307

Table 2: Comparative analysis for OASC.

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

Table 3: Results of LearnDADT.

clusters discovered against each dataset, as one can notice by looking in Table 2 at the outcomes of experiments carried out without the removal of outliers (i.e., we set $\alpha = \beta = 0$, while keeping fixed the other thresholds). In particular, we did not find clusters whose size is definitively smaller than the average in *Log A* and *Log CAD*; yet, we found various outliers over these two logs, whose removal was beneficial on the quality of the resulting models. Instead, on *Log B*, we recognized a higher number of behavioral classes, owing to the presence of small groups of (atypical) log traces.

Table 2 allows also for contrasting the proposed approach to two other methods for clustering log traces, combining a k-medoid scheme with two different measures: the edit distance [4] and the Jaccard index computed over the vectorial representation of traces proposed in [3] (precisely, a balanced combination of the "task profile" and "transition profile"). In order to apply these methods on each log, we used their respective implementations provided in ProM [14], and configured them to search for the same number of clusters as those found by OASC. In all cases, algorithm *Heuristics-Miner* [13] was exploited for inducing the workflow model of every cluster.

Table 3 shows instead some features of the *DADT* models obtained via LearnDADT on the clusters found by OASC. In particular, for each induced 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 6.1. Different settings were considered for the application of LearnDADT, which differ for the value of parameter ω (while keeping fixed $\sigma' = 0.05$), 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 available attributes refer to task elements, and there are no case attributes. In particular, we focus on two different options for setting the parameter ω:

1. $\omega = 1$, which practically makes our approach coincide with algorithm J48—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 5. This value was pragmatically chosen based on the observation that it ensured a good compromise between classification accuracy and structural conformance. However, similar results were obtained for $0.3 \leq \omega \leq 0.7$.

Note that results in Table 3 confirm that the proposed approach allows to achieve good effectiveness in all considered analysis scenarios, and that 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. Moreover, 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 of Figure 5 and classical decision-tree induction algorithms, such as C4.5 and its variant J48 [11, 39]. In this respect, 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. Instead, perturbing the attribute selection criterion with our heuristic based on task precedences produces a slight decrease in the accuracy of the model, mainly owing to the fact that additional constraints limit the selection of most predictive features. Such an effect does not arises on *Log B* and *Log CAD*, where 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 cases considering the logic of business processes can guarantee better results than inducing the classification model via the classical greedy approach, based on entropy reduction.

6.4. Experimental Results: Runtime-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 discussed in Section 1, where the behavioral cluster of a forthcoming process instance should be estimated possibly before it has been completed.

In order to conduct the analysis, we measured the accuracy of the classification model over several logs obtained from the three datasets, by including the *k*-prefix of each log traces, for *k* ranging from 1 to the maximal trace length. For *Log B* and *Log CAD*, Figure 9 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 5 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. In all cases, classification models make better

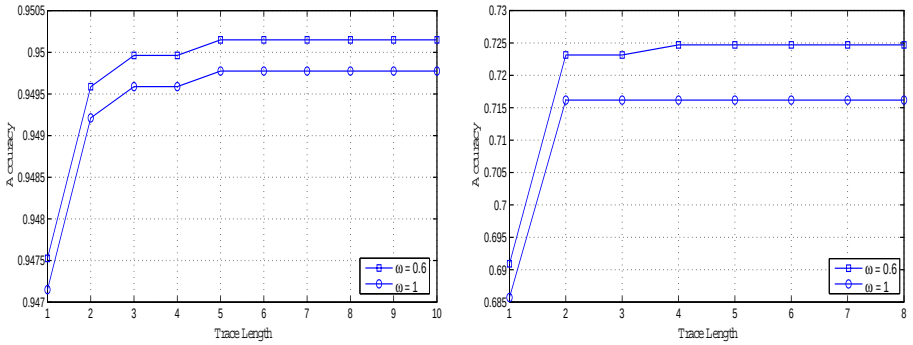


Figure 9: "On-the-fly" prediction on logs *B* (left) and *CAD* (right): accuracy vs. trace fragments' length.

predictions over longer traces, and it is encouraging to notice that our technique always guarantees more accurate results than the classical induction method.

6.5. Further Effectiveness and Sensitivity Analysis

In this section, we discuss the results of further experimental activities conducted to assess the efficacy of `OASC` and `LearnDADT`.

6.5.1. Studying the Efficacy and Sensitivity of `OASC` on Synthesized Data

Additional Quality Metrics. In order to evaluate the ability of `OASC` to recognize a given set of a-priori classes, we consider the standard *micro-averaged precision* measure [9], consisting in averaging over all the mined clusters the frequency of the majority class in each cluster, i.e., the maximal percentage of elements assigned to that mined cluster and coming from one input "true" cluster. This cluster-purity metric allows for comparing real classes and discovered clusters even when they diverge in their number of groups, as might well occur with our approach, where the number of clusters is selected automatically. Moreover, by interpreting outlier detection as a classification problem with two given classes, i.e. outliers vs. normal individuals, we will measure outlier-detection precision by computing the rates FN of False Negatives (i.e. outliers deemed as normal) and FP of False Positives (i.e. normal traces deemed as outliers), or classical measures of *Precision* (i.e. $P = TP/(TP+FP)$), *Recall* (i.e. $R = TP/(TP+FN)$) and balanced *F-measure* (i.e. $F_1 = (2 \times P \times R)/(P+R)$))—with TP denoting the number of true positives, i.e. correctly identified outliers.

Labelled Synthesized Data. Further tests on `OASC` have been conducted on syncretized data. A random generator has been implemented which produces a trace log according to four main data distribution parameters: N_A, N_T, N_C, p_C^{out}, p^{out}. More specifically, the log will contain N_T traces over an alphabet of N_A tasks, with the traces grouped in N_C clusters, and $p_C^{out} \times N_T$ of them falling into clusters whose size is smaller than the average. Additional $p^{out} \times N_T$ traces will be also generated that do not comply with any cluster at all – hence, the total percentage of outliers in the dataset is $p_C^{out} + p^{out}$. Basically, the generation process proceeds as follows. First, a set P of disjoint subschemas is built, each of which contains a number of activities randomly taken

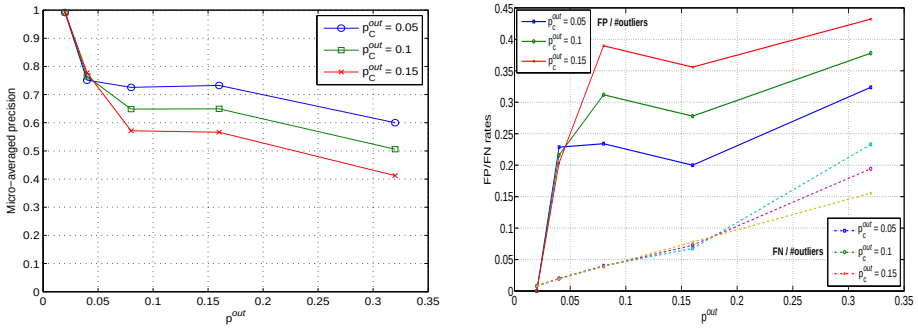


Figure 10: Sensitivity to data distribution: clustering precision (left) and outlier detection precision (right).

from a gaussian distribution with mean S_P. All these schemas are then combined into a single one W_P (with all the N_A tasks), where each sub-schema is allowed to be run independently of the others. Then, N_C subsets of P are randomly selected and enacted (according to p_C^{out}) in W_P, thereby generating the various clusters of traces over a total of $(1-p^{out}) \times N_T$ traces. Finally, $p^{out} \times N_T$ traces are generated by simulating enactments that do not comply with W_P.

Experimental Results: Precision against Data Distributions. In a first series of experiments, we generated several logs with different percentages of outliers, by varying both p^{out} (from 0.02 to 0.32) and p_C^{out} (from 0.05 to 0.15), and keeping fixed N_A=180, N_T=16000, N_C=4, and S_P=6. Figure 10 illustrates the results obtained against these data by applying algorithm OASC with γ=4, α=0.4 and β=0.5, and *pattSize*=8. When increasing the fraction of outliers in the dataset, both the precision of clusters and the ability of detecting outliers get worse. However, OASC is capable of achieving satisfactory performances, in particular if the overall percentage of outliers does not exceed 9%. In fact, under this condition the algorithm fails to detect only a very little fraction of outliers, yet producing a higher number of false positives (specially for higher values of p_C^{out}). However, since the FP counts shown in the figure are normalized w.r.t. to the total number of outliers in the dataset (and not w.r.t. to the total number of instances), only a little fraction of normal objects are purged out erroneously, and the original groups of normal clusters can be detected adequately.

Note that the low FN rates obtained by OASC is a remarkable achievement in comparison with previous anomaly detection approaches in the field of process mining. As an example, we simulated the procedure described in [30] with the help of ProM [14], by eventually extracting one single workflow model for each log, and then adopting the compliance with it as anomalousness criterion. The results in this case are notably worse than those in Figure 10, since with all data distributions most of the outlier traces are not recognized at all, thus leading to quite higher FN rates.

Experimental Results: Sensitivity to Parameters. In a second set of experiments, we measured the sensitivity of OASC to its input parameters. To this end, we generated a log by setting p^{out}=0.05 and p_C^{out}=0.05, and the same values as above for all other data parameters. In the following we discuss results of experiments performed with β=0.1

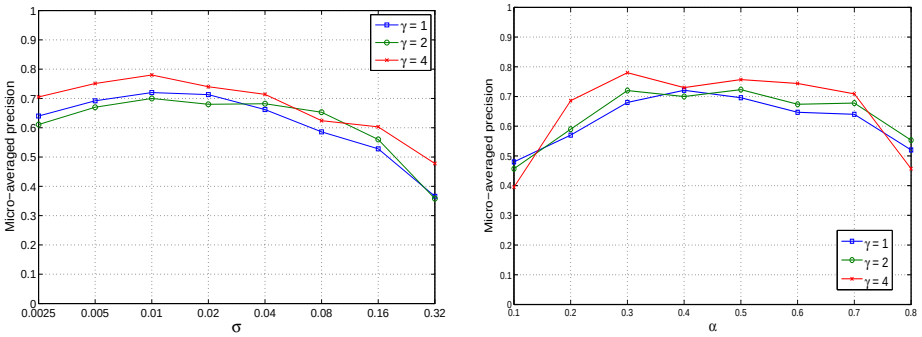


Figure 11: Clustering accuracy vs. σ and γ (left), and vs. α and γ (right)—while fixing α=0.4 and σ=0.1, resp.

and *pattSize*=8, by focusing on σ, γ and α—which appeared to require more care in finding an appropriate setting.

Figure 11 reports the micro-average precision scores computed for the clusterings discovered when using different values of σ, α, and γ. The first two parameters considerably impact the purity of the clusters. Probably, extreme values of σ yield too many (confusing) or too few patterns, which do not allow to separate the original classes into different clusters. Similarly, with extreme values of α, beside having a negative effect on the quality of pattern clusters, one increases the risk of confusing normal instances with outliers, or viceversa. Satisfactory results are obtained with σ around 0.1 and with α between 0.25 and 0.35. Moreover, using $\gamma = 4$ seems to produce additional benefits, mainly as concerns the stability of results.

Figure 12 sheds light on the ability to discriminate outliers from normal traces, for different configurations of the algorithm. As expected, recall scores tend to improve when increasing either σ or α, whereas an opposite behavior is exhibited by precision results. In fact, as discussed before, in both cases higher amounts of objects are likely to be assigned to no cluster (or to a small-sized outlier one). As to F-measure, a good trade-off seems to be found for α and σ near to 0.4 and to 0.1, respectively.

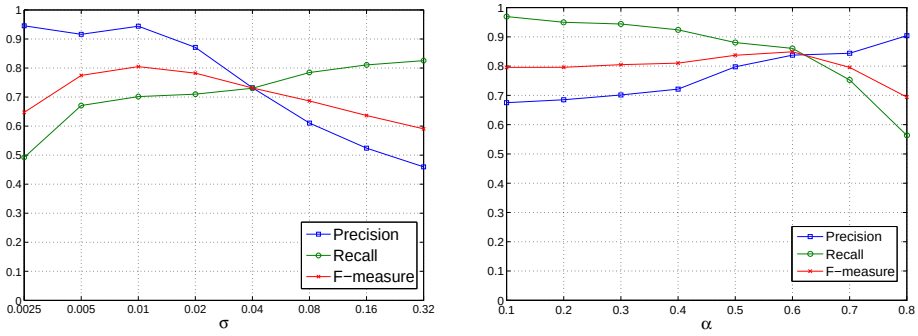


Figure 12: Outlier detection accuracy vs σ, and α—while fixing α=0.4 and σ=0.1, resp., and γ=4.

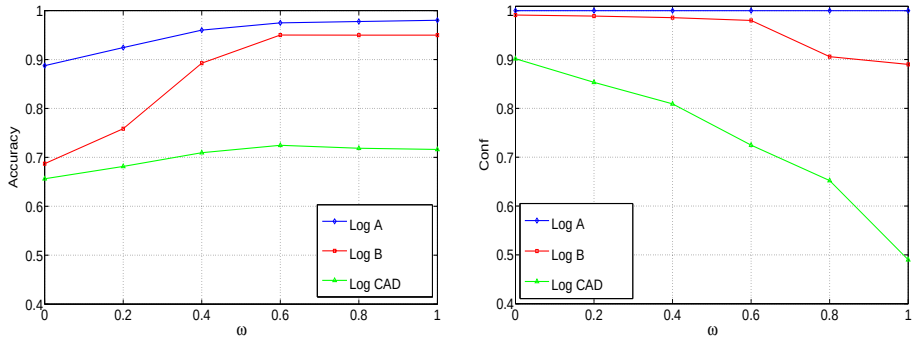


Figure 13: Sensitivity to ω (σ′=0.1): prediction accuracy (left) and conformance to task precedences (right).

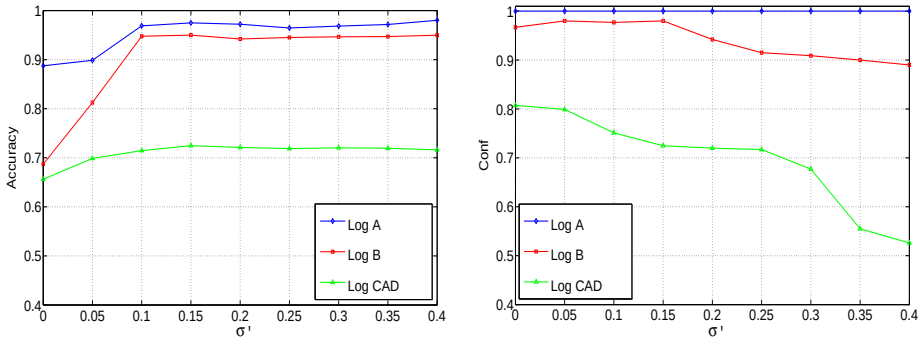


Figure 14: Sensitivity to σ′ (ω=0.6): prediction accuracy (left) and conformance to task precedences (right)

6.5.2. Sensitivity Tests on `LearnDADT`

In order to better comprehend the behavior of algorithm `LearnDADT`, we tried it with different settings directly on the three real datasets described in Section 6.2. Again, two different quality metrics of the *DADT* models were taken under consideration: prediction accuracy—w.r.t. the clusters previously discovered by `OASC` (cf. Section 6)—, and conformance to task precedences computed through measure *Conf* (Section 6.1). We next focus only on the dependence of these measures on parameters ω and σ′, and consider the results of tests performed with *minCard*=0—actually, we noticed that this parameter (blocking the expansion of nodes with too few instances), has little influence on the performances of the model, and that post-pruning techniques are enough to prevent overfitting.

The impact of ω on the metrics is illustrated in Figure 13. As expected, in correspondence of higher values of ω we can observe higher accuracy and lower conformance, and vice-versa. This is true for all the datasets, with the exception of the first, where the conformance measure is always maximal, independently of `LearnDADT` parameters—maybe due to the very simple flow models that characterize this case. In the other cases, it seems that a good trade-off is achieved around $\omega = 0.6$. Lower values, indeed, cause sensible accuracy loss. Figure 14 lets us conclude that a similar effect is produced by σ′, apart from the fact that the best trade-off between accuracy

and conformance happen for $\sigma' \in [0.1..0.15]$.

As a final remark note that further sensitivity tests were conducted on synthesized logs, with different distributions of non-structural attributes. However, since the outcomes of these tests, as far as concerns the effect of internal parameters, are substantially similar to those obtained on real logs, we omit their description here.

7. Discussion and Conclusions

In this paper, we complemented current research on clustering approaches for process mining applications, by focusing on two problems that have received little attention in earlier literature: singling out outliers from input traces, and finding predictive models for clustering results. We proposed an outlier-aware clustering method, where the similarity criterion roughly focus on the idea of correlating the traces via a special kind of frequent (concurrency-aware) structural patterns, which are preliminary discovered as an evidence of "normal" behavior. Moreover, we discussed an algorithm for decision tree learning, which is aimed at capturing the relationships between structural execution classes (possibly found by clustering) and non-structural process features. In order to predict as soon as possible the behavioral class of a novel enactment, the algorithm has been devised in a way that the sooner an attribute tends to be known along the course of process enactments, the closer it appears to the root. Encouraging results on two complex real-life application scenarios confirmed the capability of the proposed approach to discover expressive and comprehensible process models, as well as to support run-time forecasting accurately.

Comparison with related works. A first distinguishing feature of our work is its aim to induce a multi-perspective process model, capturing both typical execution scenarios and exceptional instances, as well as the links between these scenarios and non-structural context data. Indeed, our clustering approach is meant to identify both outliers and groups of normal instances, while previous proposals in the field of process mining addressed just one of these kinds of tasks, possibly using the other in an instrumental manner. In fact, in other trace clustering approaches [2–4] no attention is given to the presence of outliers, while previous efforts for mining anomalous traces [30, 31] adopt a model-based strategy, disregarding to separate multiple execution scenarios.

Technically, our approach combines the mining of advanced structural patterns with a coclustering scheme focused on the associations between such patters and the given log traces. This makes our approach neatly different from those that simply treat the log traces as symbolic sequences (e.g. [4]). Even though structural patterns were already used for clustering purposes in previous works [2, 3], our approach is different both for the kind of patterns adopted, and for its fuzzy notion of support (taking into account the interleaving of parallel branches). Moreover, differently from these works, we do not use such patterns for producing a vectorial representation of the traces, prior to the application of classic clustering methods. Rather, we look at the associations between traces and patterns themselves in a way similar to coclustering methods [9], famous for their good performances against sparse high-dimensional data. In fact, in the case of a complex process with a high number of activities (and possibly of relevant structural patterns), any vector-based approach like [2, 3] has a considerable risk to incur in

the notorious "curse of dimensionality" problem. These aspects also differentiate our work from other approaches to the detection of outliers among log traces or sequences, which either disregard the presence of different behavioral clusters [30, 31], or rely on proximity notions or on generative models (e.g., [27–29]) not taking care of the peculiar nature of process instances (e.g., intra-parallelism).

Finally, as to the induction of predictive models, we reused a classical top-down greedy scheme for the induction of a decision tree, by integrating into it the constraints needed to support on-the-fly prediction. The reasons for this choice are the readability of discovered models, the availability of noise-resistant and efficient induction algorithms, and the possibility to easily adapt the learning scheme to the prediction of workflow executions' structure, by simply correcting the attribute selection criteria with the introduction of a workflow-oriented score. Actually, our approach is independent of the particular induction algorithm and associated purity metrics adopted in this paper (*C4.5* and the *Gain Ratio*, respectively). In fact, our choice mainly served the goal of demonstrating that a pretty simple refinement of a classical induction algorithm is enough to effectively make on-the-fly prediction on the structure of a process instance.

Future work. A number of challenging issues still remain open, and are left as the subject of future research work. Firstly, we are planning to enforce the practical relevance of the discovered behavioral classes by equipping our clustering approach with the capability to take into account key process performance indicators (such as task duration, task costs, and other application-dependent QoS metrics), as well as to reuse available background knowledge on well-specified alternative usage scenarios.

Another avenue of research might be that of integrating the self-tuning techniques in [7] with our outlier detection method, as to reduce as much as possible human intervention in setting the appropriate thresholds in the mining process. More generally, based on our empirical sensitivity analysis, we notice that certain key parameters (e.g., σ, α) influence effectiveness results according to quasi-monotonic or quasi-convex curves. This behavior leaves space to the design of efficient self-tuning heuristics for a semi-automatic setting of the parameters.

Finally, in order to exploit fully the predictive power of discovered decision trees, it would be of interest to investigate the integration of such models with run-time support mechanisms (involving, e.g., task scheduling and resource allocation) of some real process management platforms.

Bibliography

[1] W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, A. J. M. M. Weijters, Workflow mining: A survey of issues and approaches, Data Knowledge Engineering 47 (2) (2003) 237–267.

[2] G. Greco, A. Guzzo, L. Pontieri, D. Saccà, Discovering Expressive Process Models by Clustering Log Traces, IEEE Transactions on Knowledge and Data Engineering 18 (8) (2006) 1010–1027.

[3] M. Song, C. W. Günther, W. van der Aalst, Trace Clustering in Process Mining, in: Business Process Management Workshops, 109–120, 2008.

[4] R. P. J. C. Bose, W. 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), 401–412, 2009.

[5] S. Subramaniam, V. Kalogeraki, D. Gunopulos, F. Casati, M. Castellanos, U. Dayal, M. Sayal, Improving process models by discovering decision points, Information Systems 32 (7) (2007) 1037–1055.

[6] A. Rozinat, W. M. P. van der Aalst, Decision Mining in ProM, in: Proc. of 4th Intl. Conf. on Business Process Management (BPM'06), 420–425, 2006.

[7] H.R.M. Nezhad, R. Saint-Paul, B. Benatallah, F. Casati, Deriving Protocol Models from Imperfect Service Conversation Logs, IEEE Transaction on Knowledge and Data Engeneering 20 (12) (2008) 1683–1698.

[8] L. Maruster, A. J. M. M. Weijters, W. M. P. van der Aalst, 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) (2006) 67–87.

[9] I. S. Dhillon, S. Mallela, D. S. Modha, Information-theoretic co-clustering, 89–98, 2003.

[10] 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, April 2002.

[11] J. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, 1993.

[12] J. R. Quinlan, Induction of Decision Trees, Machine Learning 1 (1) (1986) 81–106.

[13] A. J. M. M. Weijters, W. M. P. van der Aalst, Rediscovering Workflow Models from Event-Based Data using Little Thumb, Integrated Computer-Aided Engineering 10 (2) (2003) 151–162.

[14] B. F. van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, W. M. P. van der Aalst, The ProM Framework: A New Era in Process Mining Tool Support, in: Proc. of 26th Intl Conf on Applications and Theory of Petri Nets, 444–454, 2005.

[15] A. Rozinat, W. M. P. van der Aalst, Conformance Checking of Processes Based on Monitoring Real Behavior, Information Systems 33 (1) (2008) 64–95.

[16] P.-N. Tan, M. Steinbach, V. Kumar, Introduction to Data Mining, (First Edition), Addison-Wesley Longman Publishing Co., Inc., 2005.

[17] J. Yang, W. Wang, CLUSEQ: efficient and effective sequence clustering, in: Proc. of 19th IEEE Int. Conf. on Data Engineering (ICDE'03), 101–112, 2003.

[18] V. Barnett, T. Lewis, Outliers in Statistical Data, John Wiley, 1994.

[19] D. Hawkins, Identifications of Outliers, Chapman and Hall, 1980.

[20] E. M. Knorr, R. T. Ng, V. Tucakov, Distance-Based Outliers: Algorithms and Applications, VLDB Journal 8 (3-4) (2000) 237–253.

[21] M. M. Breunig, H.-P. Kriegel, R. T. Ng, J. Sander, LOF: Identifying Density-Based Local Outliers, in: Proc. of 2000 ACM SIGMOD International Conference on Management of Data(SIGMOD'00), 93–104, 2000.

[22] I. Gath, A. Geva, Fuzzy Clustering for the Estimation of the Parameters of the Components of Mixtures of Normal Distribution, Pattern Recognition Letters 9 (1989) 77–86.

[23] M. Jiang, S. Tseng, C. Su, Two-phase Clustering Process for Outlier Detection, Pattern Recognition Letters 22 (2001) 691–700.

[24] D. Yu, G. Sheikholeslami, A. Zhang, FindOut: Finding Outliers in Very Large Datasets, Knowledge and Information Systems 4 (4) (2002) 387–412.

[25] V. Chandola, A. Banerjee, V. Kumar, Anomaly Detection : A Survey, ACM Computing Surveys 41 (3) (2009) Article 15.

[26] S. A. Hofmeyr, S. Forrest, A. Somayaji, Intrusion detection using sequences of system calls, Journal of Computer Security 6 (3) (1998) 151–180.

[27] C. C. Michael, A. Ghosh, Two state-based approaches to program-based anomaly detection, in: Proc. of 16th Annual Computer Security Applications Conference (ACSAC'00), 21, 2000.

[28] P. Sun, S. Chawla, B. Arunasalam, Mining for Outliers in Sequential Databases, in: Proc. of 6th SIAM Int. Conf. on Data Mining, 94–104, 2006.

[29] C. Warrender, S. Forrest, B. Pearlmutter, Detecting Intrusions Using System Calls: Alternative Data Models, in: In IEEE Symposium on Security and Privacy, 133–145, 1999.

[30] F. de Lima Bezerra, J. Wainer, W. M. P. van der Aalst, Anomaly Detection Using Process Mining, in: Enterprise, Business-Process and Information Systems Modeling (BPMDS'09), 149–161, 2009.

[31] F. de Lima Bezerra, J. Wainer, Anomaly detection algorithms in business process logs, in: Proc. of 10th Int. Conf. on Enterprise Information Systems (ICEIS'08), 11–18, 2008.

[32] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning, Springer, 2003.

[33] L. Breiman, J. Friedman, R. Olshen, C. Stone, Classification and Regression Trees, Wadsworth and Brooks, 1984.

[34] D. Heath, S. Kasif, S. Salzberg, Induction of Oblique Decision Trees, Journal of Artificial Intelligence Research 2 (2) (1993) 1–32.

[35] Q. Wang, C. Suen, Large tree classifier with heuristic search and global training, IEEE Trans. Pattern Anal. Mach. Intell. 9 (1) (1987) 91–102.

[36] J. R. Quinlan, Simplifying decision trees, Int. J. Man-Machine Studies 27 (3) (1987) 221–234.

[37] M. Mehta, J. Rissanen, R. Agrawal, MDL-Based Decision Tree Pruning, in: Proc. of 1st Int. Conf. on Knowledge Discovery and Data Mining (KDD'95), 216–221, 1995.

[38] L. Rokach, O. Maimon, Top-Down Induction of Decision Trees Classifiers – A Survey, IEEE Trans. on Systems, Man and Cybernetics 35 (4) (2005) 476–487.

[39] I. H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques (2nd Ed.), Morgan Kaufmann, 2005.

[40] –, Online appendix, available at `http://si.deis.unical.it/guzzo/wfmining/appendix.pdf`