Outlier Detection Techniques for Process Mining Applications

Lucantonio Ghionna[1], Gianluigi Greco[1], Antonella Guzzo[2], and Luigi Pontieri[3]

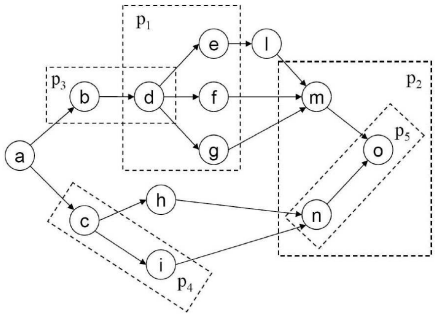
Dept. of Mathematics[1], UNICAL, Via P. Bucci 30B, 87036, Rende, Italy
DEIS[2], UNICAL, Via P. Bucci 30B, 87036, Rende, Italy
ICAR-CNR[3], Via P. Bucci 41C, 87036 Rende, Italy
{ghionna,ggreco}@mat.unical.it, {guzzo}@si.deis.unical.it,
{pontieri}@icar.cnr.it

Abstract. Classical outlier detection approaches may hardly fit process mining applications, since in these settings anomalies emerge not only as deviations from the sequence of events most often registered in the log, but also as deviations from the behavior prescribed by some (possibly unknown) process model. These issues have been faced in the paper via an approach for singling out anomalous evolutions within a set of process traces, which takes into account both statistical properties of the log and the constraints associated with the process model. The approach combines the discovery of frequent execution patterns with a cluster-based anomaly detection procedure; notably, this procedure is suited to deal with categorical data and is, hence, interesting in its own, given that outlier detection has mainly been studied on numerical domains in the literature. All the algorithms presented in the paper have been implemented and integrated into a system prototype that has been thoroughly tested to assess its scalability and effectiveness.

1 Introduction

Several efforts have recently been spent in the scientific community and in the industry to exploit data mining techniques for the analysis of process logs [12], and to extract high-quality knowledge on the actual behavior of business processes (see, e.g., [6, 3]). In a typical process mining scenario, a set of traces (registering the sequencing of activities performed along several enactments) is given to hand and the aim is to derive a model explaining all the episodes recorded in them. Eventually, the "mined" model is used to (re)design a detailed process schema, capable to support forthcoming enactments. As an example, the event log (over activities a, b, ...o) shown in the right side of Figure 1 might be given in input, and the goal would be to derive a model like the one shown in the left side, representing a simplified process schema according to the intuitive notation where precedence relationships are depicted as directed arrows between activities (e.g., b must be executed after a and concurrently with c).

In the case where no exceptional circumstances occur in enactments, process mining techniques have been proven to discover accurate process models. However, logs often reflect temporary malfunctions and evolution anomalies (e.g., traces $s_9, ..., s_{14}$ in the example above), whose understanding may help recognizing critical points in the process that could yield invalid or inappropriate behavior.



s_1 : acbdhefgnlmo (3)		s_8 : acino (20)	
s_2 : abchdefgnlmo (3)		s_9 : abcd (2)	
s_3 : abchdgfelmno (2)		s_{10} : acmo (1)	
s_4 : acbhdfenglmo (2)		s_{11} : acc (2)	
s_5 : acbdhnegflmo (1)		s_{12} : acon (2)	
s_6 : abcideghnflmo (2)		s_{13} : abe (1)	
s_7 : acbdfgheilmno (1)		s_{14} : acdo (2)	

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

In the paper, this peculiar aspect of process mining is investigated and the problem of singling out exceptional individuals (usually referred to as *outliers* in the literature) from a set of traces is addressed.

Outlier detection has already found important applications in bioinformatics [1], fraud detection [5], and intrusion detection [9], just to cite a few. When adapting these approaches for process mining applications, novel challenges however come into play:

(C1) On the one hand, looking only at the sequencing of the events may be misleading in some cases. Indeed, real processes usually allow for a high degree of concurrency, and are to produce a lot of traces that only differ in the ordering between parallel tasks. Consequently, the mere application of existing outlier detection approaches for sequential data to process logs may yield many *false positives*, as a notable fraction of task sequences might have very low frequency in the log. As an example, in Figure 1, each of the traces in $\{s_1, ..., s_5\}$ rarely occurs in the log, but it is not to be classified as anomalous. Indeed, they correspond to a different interleaving of the same enactment, which occurs in 10 of 40 traces.

(C2) On the other hand, considering the compliance with an ideal schema may lead to *false negatives*, as some trace might well be supported by a model, yet representing a behavior that deviates from that observed in the majority of the traces. As an example, in Figure 1, traces s_6 and s_7 correspond to the same behavior where all the activities have been executed. Even though this behavior is admitted by the process model on the left, it is anomalous since it only characterizes 3 of 40 traces.

Facing **(C1)** and **(C2)** is complicated by the fact that the process model underlying a given set of traces is generally unknown and must be inferred from the data itself. E.g., in our running example, a preliminary question is how we can recognize the abnormality of $s_9, ..., s_{14}$, without any a-priori knowledge about the model for the given process.

Addressing this question and subsequently **(C1)** and **(C2)** is precisely the aim the paper, where an outlier detection technique tailored for process mining applications is discussed. In a nutshell, rather than extracting a model that accurately describes all possible execution paths for the process (but, the anomalies as well), the idea is of capturing the "normal" behavior of the process by simpler (partial) models consisting of *frequent structural patterns*. More precisely, outliers are found by a two-steps approach:

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

By this way, we will discover, e.g., that traces $s_9, ..., s_{14}$ do not follow any of the frequent behaviors registered in the log. Moreover, we will reduce the risk of both false positives (traces are compared according to behavioral patterns rather than to the pure sequencing of activities) and false negatives (traces that comply with the model might be seen as outliers, if they correspond to unfrequent behavior)—cf. **(C1)** and **(C2)**.

Organization. The above techniques are illustrated in more details in Section 2, while basic algorithmic issues are discussed in Section 3. After illustrating experimental results in Section 4, we draw some concluding remarks in Section 5.

2 Formal Framework

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

Behavioral Patterns over Process Logs. The first step for detecting outliers is to characterize the "normal" behavior registered in a process log. In the literature, this is generally done by assessing the causal relationships holding between pairs of activities (e.g., [3, 10]). However, this does not suffice to our aims, as abnormality may emerge not only w.r.t. the sequencing of activities, but also w.r.t. constructs such as branching and synchronization. Hence, towards a richer view of process behaviors, we next focus on the identification of those features that emerge as complex patterns of executions.

Definition 1 (S-Pattern). A *structural* pattern (short: S-pattern) over a given set A of activities is a graph $p = \langle A_p, E_p \rangle$, with $A_p = \{n, n_1, \ldots n_k\} \subseteq A$ such that either: **(a)** $E_p = \{n\} \times (\{n_1, \ldots n_k\})$ – in this case, p is called a $FORK$-pattern–, or **(b)** $E_p = (\{n_1, \ldots n_k\}) \times \{n\}$ – in this case, p is called a $JOIN$-pattern. Moreover, the *size* of p, denoted by $size(p)$, is the cardinality of E_p. □

In particular, an S-pattern with size 1 is both a $FORK$-pattern and a $JOIN$-pattern, and simply models a causal precedence between two activities. This is, for instance, the

case of patterns p_3, p_4, and p_5 in Figure 1. Instead, higher size patterns account for fork and join constructs, which are typically meant to express parallel execution (cf. p_1) and synchronization (cf. p_2), respectively, within concurrent processes. The crucial question is now to formalize the way in which patterns emerge for process logs.

Definition 2 (Pattern Support). Let t be a trace and $p = \langle A_p, E_p \rangle$ be an S-pattern. We say that t *complies with* p, if **(a)** t includes all the activities in A_p and **(b)** the projection of t over A_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:

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

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

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

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

In addition, differently from many pattern mining approaches, the frequency of a pattern p does not necessarily indicate its relevance to modeling the process behavior. In particular, when comparing two σ-frequent patterns p and p' such that p is a subgraph of p', we can safely focus on p' if its frequency does not differ significantly from that of p; otherwise, i.e., if p is much more frequent than p', the subpattern p is also interesting its own, as it can help recognizing relevant behavioral clusters. This is formalized below.

Definition 3 (Interesting Patterns). Let L be a log, and σ, γ be two real numbers. Given two S-patterns p and p', we say that p' *γ-subsumes* p, denoted by $p \sqsubseteq_\gamma p'$, if p is a subgraph of p' and $supp(p, L) - supp(p', L) < \gamma \cdot supp(p', L)$. Further, 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' over A s.t. $size(p')$=$size(p) + 1$, p' is σ-frequent on L, and $p \sqsubseteq_\gamma p'$. □

Cluster-based Outliers. Once that "normality" has been modeled by means of the discovery of interesting patterns, we can then look for those individuals whose behavior deviates from the normal one. To this end, the second step of our outlier detection approach performs a "co-clustering" (see, e.g., [2]) of patterns and traces, based on their

mutual correlation captured by the $supp$ measure. Intuitively, we look for associating pattern clusters with trace clusters, so that outliers emerge as those individuals that are not associated to any pattern cluster or that belong to clusters whose size is definitively smaller than the average cluster size. Abstracting from the specificity of the mining algorithm (discussed in Section 3), the output of this method is formalized below.

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

- $\widehat{P}$=$\{\widehat{p}_1, ..., \widehat{p}_k\}$ is a set of non-empty P's subsets (named *pattern clusters*) s.t. $\bigcup_{j=1}^{k} \widehat{p}_j = P$;
- $\widehat{L}$=$\{\widehat{l}_1, ..., \widehat{l}_h\}$ is a set of non-empty disjoint L's subsets (named *trace clusters*) such that $\bigcup_{i=1}^{h} \widehat{l}_i$ = $\{t \in L \mid \exists \widehat{p}_i \in \widehat{P}$ s.t. $supp(\widehat{p}_i, t) \geq \alpha\}$;
- $\mathcal{M} : \widehat{P} \mapsto \widehat{L}$ is an invertible function that associates each pattern cluster $\widehat{p}_j$ with a trace cluster $\widehat{l}_i$ and vice-versa, i.e., $\widehat{l}_i = \mathcal{M}(\widehat{p}_j)$ and $\widehat{p}_j = \mathcal{M}^{-1}(\widehat{l}_i)$.

Moreover, given two real numbers α, β in $[0..1]$, a trace $t \in L$ is an (α, β)-*outlier* w.r.t. C if either **(a)** $t \notin \bigcup_{i=1}^{h} \widehat{l}_i$, or **(b)** $|\widehat{l}_i| < \beta \times \frac{1}{h} \sum_{\widehat{l}_j \in \widehat{L}} |\widehat{l}_j|$, where $t \in \widehat{l}_i$. □

In words, we define outliers according to a number of clusters, discovered for both traces and patterns based on their mutual correlations, which represent different behavioral classes. More specifically, two different kinds of outlier emerge; indeed, condition **(a)** deems as outlier any trace that is not assigned to any cluster (according to the minimum support α), while condition **(b)** estimates as outliers all the traces falling into small clusters (smaller than a fraction β of the average clusters' size).

3 An Algorithm for Detecting Outliers

In this section, we discuss the `TraceOutlierMining` algorithm that discovers a set of outliers, based on the computation scheme and the framework described so far. The algorithm is shown in Figure 2: Given a log L, a natural number $pattSize$ and four real thresholds σ,γ, α and β, it first computes a set P of (σ, γ)-maximal S-patterns via the function `FindPatterns`, while restricting the search to patterns with no more than $pattSize$ arcs. Then, an α-coclustering for L and P is extracted with the function `FindCoClusters` (Step 2). The remaining 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 all traces. Clearly, the main computation efforts hinge on the functions `FindPatterns` and `FindCoClusters`, which are thus thoroughly discussed next.

Function `FindPatterns`. The main task when mining (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` uses a relaxed notion of support (denoted by $supp'$) which optimistically decreases the counting of spurious activities by a "bonus"

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
Method: Perform the following steps:

1 $P :=$ `FindPatterns` (L,$pattSize$,σ);
2 $\langle \widehat{P}, \widehat{L}, \mathcal{M} \rangle :=$ `FindCoClusters`(L,P,α);
3 $U := \emptyset$; $\quad avgSize := \frac{1}{|\widehat{L}|} \sum_{\widehat{l}_j \in \widehat{L}} |\widehat{l}_j|$;
4 **for each** trace t in L **do**
5 **if** $t \notin \bigcup_{i=1}^{h} \widehat{l}_i$, **or** $|\widehat{l}_i| < \beta \times \frac{1}{h} \sum_{\widehat{l}_j \in \widehat{L}} |\widehat{l}_j|$, where $t \in \widehat{l}_i$ **then** $U{:=}U \cup \{t\}$;
6 **return** U;

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

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

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

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

Fig. 2. Algorithm `TraceOutlierMining`

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

It can be shown that function $supp'$ is both anti-monotonic and "safe", in that it does not underestimate the real support of candidate patterns. We can hence exploit it to implement a level-wise search of patterns: After building (in Step P1) the basic set L_1 of frequent S-patterns with size 1 (i.e., frequent activity pairs), an iterative scheme is used to incrementally compute any other set L_k, for increasing pattern size k (Steps P4–P8), until either no more patterns are generated or k reaches the upper bound given as input. In more detail, for each $k > 1$, the function $generateCandidates$ is first used to produce the set $Cand_k$ of k-sized candidate patterns, by suitably extending the patterns in L_{k-1} with those in L_1 (Step P4). The set L_k is then filled with the candidate patterns in $Cand_k$ that really get an adequate support in the log (Steps P5- P6). By construction of $supp'$, L_k is guaranteed to include (at least) all σ-frequent S-patterns with size k.

Eventually, by a straightforward application of Definition 3 to the patterns in L_{k-1} and L_k, all (σ, γ)-maximal S-patterns with size $k-1$ are correctly extracted and added to the set R, the ultimate outcome of `FindPatterns`. In fact, in Step P7 the original function $supp$ is actually used for checking $(\sigma.\gamma)$-maximality.

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

At start, a preliminary partition P^* of P is built by applying a clustering procedure to a similarity matrix S for P, where the similarity between two patterns roughly estimates the likelihood that they occur in the same trace. Precisely, similarity values are computed (Step C1) by regarding $supp$ as a sort of 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, many classical clustering algorithms could be used to extract P^* out of the matrix M (Step C2). In fact, we used an enhanced implementation of the *Markov Cluster Algorithm*, which achieved good results on several large datasets [4], and choose the number of clusters autonomously.

In the second phase (Steps C3-C13), the preliminary clustering P^* 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 $\widehat{p}^t$, which is the union of all the (basic) clusters in P^* that are correlated enough to t, still based on the function $supp$ and threshold α. It may happen that the cluster $\widehat{p}^t$ is already in $\widehat{P}$, for it was induced by some other traces; in this case we retrieve, by using the mapping $\mathcal{M}$, the cluster $\widehat{l}^t$ 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 $\widehat{L}$, and update $\mathcal{M}$ to store the association between this new cluster and $\widehat{p}^t$, which is stored as well, in $\widehat{P}$, as a novel pattern cluster (Step C10).

We pinpoint that algorithm `TraceOutlierMining` can be implemented without importing the entire input log in main memory, as we may just scan it k times to find the patterns, plus two further times to build the matrix M and map the traces to the clusters (Steps C4-C12). Thus, main memory computation is just limited to the clustering of interesting patterns, whose overall size can be kept low by suitably setting the thresholds σ and γ (cf. Def. 3). This can ensure scalability over huge datasets.

4 Experimental Results

The proposed approach has been implemented in a Java prototype system, and thoroughly tested to assess its scalability and accuracy, on a 1800MHz/2GB Pentium IV machine running *Windows XP Pro*. To this aim, we developed a generator that randomly produces a process log of N_T traces, by enabling to control several data distribution features. Traces in the log are distributed along N_C clusters, so that $p_C^{out} \times N_T$ of them fall into clusters whose size is smaller than the average. In addition, $p^{out} \times N_T$ traces

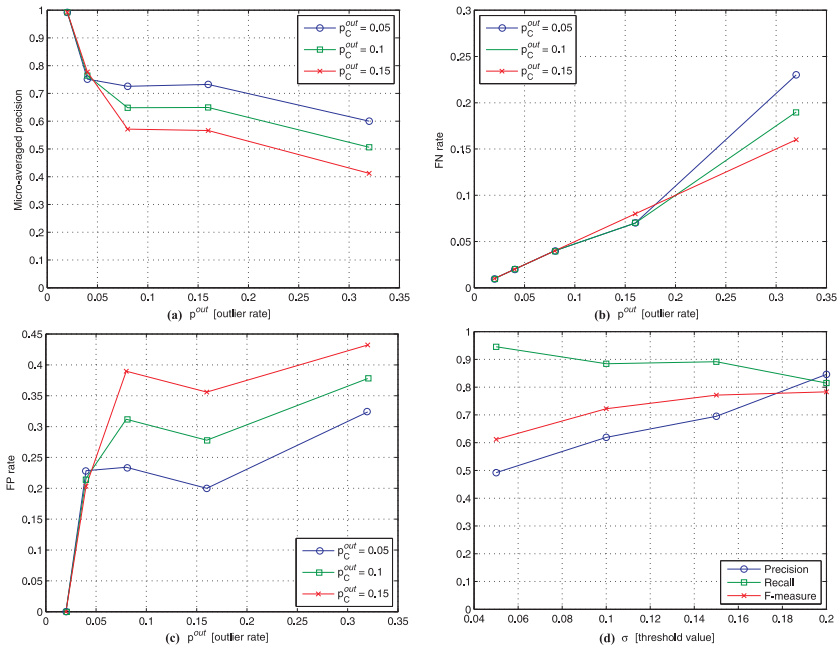


Fig. 3. Accuracy Results.

are produced that do not comply with any of the clusters. Hence, the total percentage of outliers in the dataset is $p_C^{out} + p^{out}$. In a nutshell, the generator works as follows: First, it builds a set P of disjoint subschemas whose sizes are taken from a gaussian distribution with mean S_P, and then combines them into a schema W_P over N_A activities, where each sub-schema is allowed to be run independently. 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.

Accuracy. Firstly, we evaluated the effectiveness of the approach against various input logs, containing different percentage of outliers. To this purpose, logs were generated by varying both p^{out} (from 0.02 to 0.32) and p_C^{out} (from 0.05 to 0.15), and using fixed values for the other parameters: N_A=180, N_T=16000, N_C=4, and S_P=6. Figures 3.(a), 3.(b), and 3.(c) illustrate accuracy results obtained on these data, by applying the `TraceOutlierMining` algorithm with γ=4, α=0.4 and β=0.5, and $pattSize$=8. More precisely, Figure 3.(a) depicts the accuracy of the approach in rediscovering all the clusters in the input log, according to the standard *micro-averaged precision* measure—computed by 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). Instead, Figures 3.(b) and 3.(c) depict the capability of the approach to correctly recognize anomalous traces, by reporting the rates of False Negatives (FN), i.e., outliers deemed as normal, and False Positives (FP), i.e., normal traces deemed as outliers, resp.—in a sense, the outlier detection problem is regarded here as a classification problem with two classes of objects: outliers and nor-

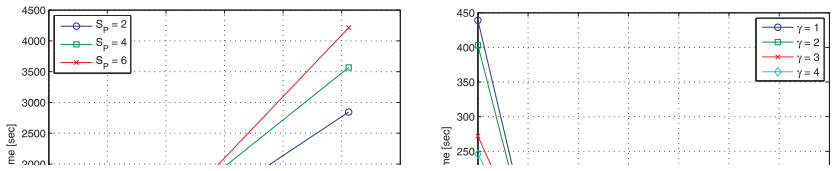


Fig. 4. Scalability Results.

mal individuals. These quality measures worsen when increasing the overall percentage of outliers, still getting quasi-optimal values when this latter is under 9%.

In order to evaluate the sensitivity of the algorithm to its parameters, we generated a log with p^{out}=0.05 and p_C^{out}=0.09, and the same value as in the previous test for any other data parameter. Figure 3.(d) reports three standard accuracy measures (namely, precision, recall, and F-measure), computed again for the binary classification problem outliers vs. normal individuals, obtained when varying σ, and for α=0.8, γ=4, β=0.5 and $pattSize$=8. The figure evidences a trade-off between precision and recall, thereby suggesting that parameters must be chosen depending on the application needs, possibly with the help of self-tuning heuristics (as in [11]).

Scalability. In another series of experiments, we assessed the scalability of the approach w.r.t. the size of the input data, by building a number of datasets with increasingly larger number of traces and activities in the schema, while fixing p^{out}=0.02, N_A=125, and N_C=4. Fixed values were taken as well for all `TraceOutlierMining`'s parameters: α=0.2, β=0.5, γ=4 and σ=0.15, and $pattSize$=10. As shown in Figure 4.(a), the total computation time linearly scales both with the number of log traces N_T and with the size of the process schema W_P used to generate the log itself (cf. S_P).

Finally, Figure 4.(b) reports the total computation time spent by the algorithm over the log for Figures 3.(a), 3.(b), and 3.(c), when keeping fixed all the parameters but σ and γ (actually, we set again α=0.8, γ=4, β=0.5 and $pattSize$=8). Note that σ and γ do impact on computation time: the lower their value the higher the time. However, a notable increase only occurs when σ passes from 0.1 to 0.05. This effect is emphasized when γ too is kept low, and any σ-frequent pattern is also (σ, γ)-maximal.

5 Discussion and Conclusion

Even though singling out anomalies in process executions can help in recognizing critical points in the process, current process mining approaches adopt very simple and pragmatical solutions in its facing. Indeed, the general idea (e.g., [10]) is to exploit user-defined thresholds to define the minimum frequency for activities below which an execution is considered noisy. Only very recently, [11] embarked on a systematic investigation of noisy environments, by focusing on the mining of conversation logs

and by proposing automatic techniques for identifying the right threshold value. In this paper, we have expanded this research line, by devising an *outlier detection* approach specifically tailored for process models and logs, which, differently from [11], takes account for concurrency constructs. Moreover, due to our focus on outlier detection rather than on noise filtering, anomalies are defined not only w.r.t. statistical global features, but also w.r.t. major behavioral clusters. In fact, clustering-based outlier detection techniques have already been used in different contexts (see, e.g., [8, 13, 9]). Here, this methodology has been applied to process models and specialized to deal with categorical data (cf. the various behavioral patterns), by sharing the perspective of [7] where outliers are characterized in terms of infrequent patterns.

The proposed approach paves the way for further elaborations. For example, an avenue of research is to integrate it with the self-tuning techniques in [11], as to reduce as much as possible human intervention in the mining process. Also, it would be relevant to investigate on the definition of explanation techniques, i.e., on methods that, given a set of outliers, abductively formulate hypotheses for recognizing critical points in the process that can yield invalid or inappropriate behavior.

References

1. A. Apostolico, M. E. Bock, S. Lonardi, and X. Xu. Efficient detection of unusual words. *Journal of Computational Biology*, 7(1/2):71–94, January 2000.
2. I. S. Dhillon, S. Mallela, and D. S. Modha. Information-theoretic co-clustering. In *Proc. 9th ACM SIGKDD Conf. on Knowledge Discovery and Data Mining (KDD03)*, 89–98, 2003.
3. S. Dustdar, T. Hoffmann, and W. M. P. van der Aalst. Mining of ad-hoc business processes with teamlog. *Data and Knowledge Engineering*, 55(2):129–158, 2005.
4. 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.
5. T. E. Fawcett and F. Provost. Fraud detection. In *Handbook of data mining and knowledge discovery*, pp. 726–731. Oxford University Press, 2002.
6. G. Greco, A. Guzzo, L. Pontieri, and D. Saccà. Discovering expressive process models by clustering log traces. *IEEE Trans. on Knowledge and Data Engin.*, 18(8):1010–1027, 2006.
7. Z. He, Z. Xu, J.Z. Huang, and S. Deng. Fp-outlier: Frequent pattern based outlier detection. In *Proc. of CIS'05*, pp. 735–740, 2005.
8. M. F. Jaing, S. S. Tseng, and C. M. Su. Two-phase clustering process for outliers detection. *Pattern Recogn. Lett.*, 22(6-7):691–700, 2001.
9. S. Jiang, X. Song, H. Wang, J.-J. Han, and Q.-H. Li. A clustering-based method for unsupervised intrusion detections. *Pattern Recogn. Lett.*, 27(7):802–810, 2006.
10. L. Maruster, A.J.M.M. Weijters, W. M. P. van der Aalst, and A. van den Bosch. A rule-based approach for process discovery: Dealing with noise and imbalance in process logs. *Data Mining and Knowledge Discovery*, 13(1):67–87, 2006.
11. H.R. Motahari Nezhad, R. Saint-Paul, B. Benatallah, and F. Casati. Protocol discovery from imperfect service interaction logs. In *Proc. of ICDE'07*, pp. 1405–1409, 2007.
12. W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. Weijters. Workflow mining: a survey of issues and approaches. *Data & Know. Engin.*, 47(2):237–267, 2003.
13. D. Yu, G. Sheikholeslami, and A. Zhang. Findout: finding outliers in very large datasets. *Knowledge Information Systems*, 4(4):387–412, 2002.