# Algorithms for Anomaly Detection of Traces in Logs of Process Aware Information Systems

Fábio Bezerra, Jacques Wainer

*Cyberspace Institute - UFRA*
*Av. Presidente Tancredo Neves, 2501*
*Belém, Pará, Brazil*

*Institute of Computing - UNICAMP*
*Av. Albert Einstein, 1251*
*Campinas, São Paulo, Brazil*

**Abstract**

This paper discusses four algorithms for detecting anomalies in logs of process aware systems. One of the algorithms only marks as potential anomalies traces that are infrequent in the log. The other three algorithms: threshold, iterative and sampling are based on mining a process model from the log, or a subset of it. The algorithms were evaluated on a set of 1500 artificial logs, with different profiles on the number of anomalous traces and the number of times each anomalous traces was present in the log. The sampling algorithm proved to be the most effective solution. We also applied the algorithm to a real log, and compared the resulting detected anomalous traces with the ones detected by a different procedure that relies on manual choices.

*Keywords:* anomaly detection, process mining, process-aware systems

## 1. Introduction and motivation

Process aware information systems (PAIS) are "a software system that manages and executes operational processes involving people, applications, and/or information sources on the basis of process models" [16]. In this paper we are interested in systems in which the execution of said processes are not predefined beforehand. Such systems fall within the ad-hoc and loosely framed systems described in[16]. The central aspect of such loosely framed system is that the people that are executing the activities are in control to decide how the case will proceed next, in order to achieve the goals for the case. Such loosely framed

*Email addresses:* fabio.bezerra@ufra.edu.br (Fábio Bezerra), wainer@ic.unicamp.br (Jacques Wainer)

[1]Institutional address for contacting Fábio Bezerra.

[2]Institutional address for contacting Jacques Wainer.

system include flexible workflow systems, case handling systems, and scientific workflows.

These system allow for more control by the people executing the process, and are a possible solution to what has been called the inflexibility of workflow systems [24, 32]. But such flexibility may come with a cost. Systems that are not under control of a pre-specified process model may be subject to frauds and errors. Detecting such cases of frauds, exceptions and errors, which we will call *anomalies*, is the goal of this research.

From the point of view of this research, the execution of a case or an instance of a process is a sequence of activities that were executed on the behalf of that case. Thus the case "the firing of John Jacob Jingleheimer Schmidt" is a instance of a process of "firing", and for Mr. Schmidt case the following activities were executed: "inform Mr. Schmidt", "calculate balance due", "explain severance benefits" and so on. In this paper, activities are considered atomic and their duration is not important, thus the set of activities executed can be seen as a *sequence*. Furthermore we will not attribute meaningful names to the activities, but refer to them using single letter names. Thus, Mr Schmidt firing case is seen as the sequence of activities *abcbd*, for example. Such sequences of single letter activities are called *traces*. The set (or better the multiset) of traces from which one is trying to identify the anomalies is called a *log*. Each trace can appear many times in the log, and thus the multiset, and each time a particular trace appears in the log is called a *trance-instance*.

This research presents results in detecting anomalies in logs of execution of PAIS, where the anomaly is detected solely based on the sequence and choices of activities that took place in that anomalous execution. Thus, using the example above, one would detect that Mr Schmidt firing was anomalous because the particular sequence *abcbd* of activities was too different from the sequences of activities for all or most of the other firing cases. For example, it may be the case that the activity "terminate Mr Schmidt system access" was performed much later than usual, which could indicate either that the system administrator was not properly trained regarding the security policies, or that there was a collusion to allow Mr Schmidt access to data he no longer should access.

Of course, the anomalous nature of a case may be derived from the values involved in some of the activities, (for example, Mr Schmidt's health benefits remain active for 300 month after his firing), or because of the people who executed some of the activities (for example, the system access termination activity was executed by a senior vice president), or because of time to perform an activity or the whole process was greater or less than normal (for example, the calculation of balance due was faster then normal). We call these examples as *data*, *organizational*, and *time* anomalies, to match the other four aspects of process models [35]. This research is restricted to *control flow* anomalies.

### *1.1. Towards a definition of anomalous trace*

Anomalous traces, once discovered, must be analyzed to find out if indeed they are examples of incorrect executions or if they are acceptable executions, and if they are found to be incorrect executions, the reasons for and consequences

of these executions must be further investigated. Thus, the algorithms discussed in this paper must be used as a first automated step towards a more comprehensive security auditing practice for flexible or loosely framed PAIS. This places some constraints for the algorithms. If we focus on a fraud perspective - that is, that the anomalous traces are a possible indication of frauds, then missing any of the potential fraudulent executions has serious consequences. Thus, the algorithms to detect the anomalous traces must have a very low false negative rate. The false negative cases are traces that the algorithm flagged as negative (or "normal") and that attribution was wrong. Such false negative cases will not be forwarded to the specialists that would determine that the trace was indeed a fraud and take the appropriate measures. On the other hand, given that this human analysis of whether an anomalous trace is indeed a fraud is a costly process, one would also prefer if the algorithms had low false positive rates - that is, the number of cases that are mistakenly flagged as anomalous when in fact they are not - should also be kept low. But a low false positive rate is less important than a low false negative rate.

If the anomalous traces are interpreted as errors, either erroneous execution or erroneous logging of the processes, then the unbalance of costs between a false negative and a false positive is less severe. A false negative will not generate the loss of revenue that an undetected fraud usually will incur. Therefore under this perspective a more even balance between false negative and false positive rates should be aimed at. In this paper we will also explore this alternative.

Finally, let us address the issue of what is an anomalous execution of a process. Chandola *et al* [9] in an important survey on anomaly detection, discuss that there is no formal definition of anomaly, only intuitions that guide the development of different algorithms and techniques. For example, one may have the intuition that "normal" data falls "together" (in some appropriate distance metric) and that anomalies are "spread appart". This intuition based on distance leads to the development of many algorithms based on nearest neighbor [9, Section 5]. If on the other hand, one has the intuition that anomalies are data points that have low probability of occurring (given the appropriate generative model for the "normal" data), this intuition leads to the development of family of techniques described in [9, Section 7] as statistical detection models.

The same apply to our research: we have no formal definition of an anomalous traces, but we have some intuitions that guided the development of the algorithms discussed herein. They are:

- the set of executions can be partitioned into a set of **normal** and **anomalous** executions
- each of the anomalous execution is "infrequent" among the set of all executions, although the whole set of anomalous executions may not be that infrequent.
- the process models that "explain" the executions in the normal set "make sense"

- the process models that could explain both the normal executions and some of the anomalous ones "make less sense"

The terms "infrequent", "explain" and "make sense" need to be further refined if one wants to transform these intuitions into one or more algorithms. Nevertheless these intuitions can be formalized in some more precise notation, leaving the uncertainties confined into a few constants and relations.

- given a set $A$ of activity names
- a trace $t$ is defined as $t \in A^*$
- a log $L$ is defined as a multiset of traces $L = \{\langle t, n_t \rangle\}$ where $n_t$ is the multiplicity of the trace in the log
- the size of a log $L$ is the number of trace-instances in it, that is $\forall \langle\, t, n_t \rangle \in L$, $\text{size}(L) = \sum n_t$
- the frequency of a trace $t$ in the log $L$ is defined as $\text{freq}_L(t) = \frac{n_t}{\text{size}(L)}$
- there exists a constant $\text{freq}_{max}$ represents the term "infrequent",
- there exists a relation "explain" between a process model $M$ and a log $L$ denoted by $M \vdash L$
- there exists a partial order "make more sense" between models denoted as $M_1 \succ M_2$ which indicates that $M_1$ "make more sense" than $M_2$

Now, our intuitions regarding anomalies can be pseudo-formalized as, given a log $L$

- $L$ can be partitioned into two multisets $A$ (anomalous) and $N$ (normal) such that $A \cup N = L$ and $A \cap N = \emptyset$
- $\forall \langle\, a, n_a \rangle \in A$, $\text{freq}_L(a) \leq \text{freq}_{max}$
- let $M_N$ be the maximum under the partial order $\succ$ of $\{M | M \vdash N\}$
- and let $M_L$ be the maximum under the partial order $\succ$ of $\{M | M \vdash L\}$
- then $M_N \succ M_L$

*1.2. Naive detection approach*

Before discussing these terms ("infrequent", "explain", and "make sense"), the intuitions above lend themselves to a first algorithm, which we call the **naive algorithm**. The naive algorithm resolves the problem of further defining the terms "explain" and "make sense" (or the relations $\vdash$ and $\succ$) by avoiding them altogether. Using only the first two points in our intuition, the naive algorithm would select as anomalous all traces that are infrequent in the log. This algorithm errs to the side of caution, that is, it will possibly flag a normal trace that happens to be infrequent as anomalous, but will not miss any anomalous traces.

This is a reasonable solution to the anomaly detection problem because as we discussed it results in zero false negative rate, which is the hard requirement for the algorithms. But likely, the naive algorithm will result in the largest false positive rate. Thus all algorithms must be evaluated against the naive one, since it satisfy the hard requirement of the problem.

*1.3. Process mining and analysis*

Let us now go back to the terms "explain" and "make sense". For this paper, a process model "explains" a set of executions if the model was mined from the executions. The literature on proces mining will be reviewed in Section 4, but for this introduction it suffices to say that process mining are heuristic algorithms that generate a process model from logs, so that all or most of the traces are *instances of the model*, that is, they are the result of following a particular path in the process model.

As discussed in [43], process mining is not a well defined problem, since there are infinite process models that can generate a particular set of traces, and thus each mining algorithm searches and finds a particular solution in this space of possible process models.

In a broad terms, process mining algorithms fall into two distinct categories, the ones that result in a process model that can generate **all** traces from which it was mined, which we will call a *precise mining*, and the ones that only generate **part** of the traces, which we will call a *noisy mining*. The latter group would accept not to generate a few particular traces from the log if it considers that including those traces would make the resulting process model too complex or too general. In this case, such algorithms would consider the traces not included in the mined process as *noise*.

But the noise traces are not totally unconstrained: they must be at least partially supported by the mined model, that is, the mined model must be able to generate most of each of the noise traces, but some of the activities in each of the noise traces may be incorrect (from the point of view of the mined model). Thus one may define metrics that relates a trace with a process model, usually referred as *conformance* of the trace in relation to the model, which measures how much of the trace is generated by the model. A conformance of one indicates that the trace is fully generated by the model, and thus it is an instance of the model.

The "make sense" term seems to refer to a semantic aspect of the model and may require human analysis. For example, we know that it "makes little sense" to perform a request activity and approval of the request activity in parallel. Further information on security constraints ("all requests must be *followed* by a corresponding approval"), or data dependencies ("the approval activity uses the data generated by the request activity") could indicate that the model "make little sense", but such information is not available for the anomaly detector. But there are other aspects of "make sense" that could be available to the system. They refer either to the structure of the model itself, regardless of what the activities in it mean, or refer to the relation between the model and the traces used to mine it. For example, regarding solely the model structure,

one may define a "make more sense than" relation among models that include components such as:

- smaller models, that is, models with less connectives and activities, are preferred to larger models;
- models with less repetition of activities are preferred to the ones with more repetition.

The "make more sense than" relation can be then materialized into a metric for the model structure. In fact there is a growing research in process model metrics [42]. The idea of ordering process models regarding on how much they "make sense" is called *appropriateness* by [29] and it is measured using *behavioral* and *structural* perspectives.

Regarding the relation between the model and the log it was mined from, one may prefer models that are neither too specific or too general in relation to the log. The terms *overfit* and *underfit* have also been used to refer to models that are too specific or too general. A model is *too specific* if the set of execution traces it can generate and the set of traces in the log are exactly the same. This by itself is not a problem as the term overfitting would suggest for the reader familiar with the machine learning literature.

In machine learning, overfitting refers to models that are too specific to the data they were trained on and thus have accuracy for classifying *new* data lower than expected. Thus in machine learning, overfitted models do not generalize well. This is not exactly the problem in process mining, since there are no *new* traces to classify as either being or not an instance of the mined process. The problem is that the most specific (precise) model that can be mined from a set of traces is an OR of each of the traces, and that model will "make less sense" from a structural point of view, because it is too large and it has too much duplication of activities.

A model underfits the log if there are too many traces allowed in the model that are not present in the log; in this case the model is said to be *too general.* But this comparison cannot be naively performed - a process model that contains a loop will generate an infinite number of traces, and the log must be necessarily finite, and thus there will always be too many possible execution paths in the model that are not represented in the log.

There is no agreed upon definition of what is exactly the correct balance between specificity or generality, and each mining algorithm will implement either implicitly (as most do) or explicitly a particular balance between the two extremes of specificity and generality. More precisely, there is an usually implicit partial order $\succ$ between not just process models by itself, but between a pair of a model and log from which it was mined, that is, $\langle M_1, L\rangle \succ \langle M_2, L\rangle$, and the mining algorithm will search for the model that will result in one of the maximum for a particular log.

We can now return to the problem of detecting anomalies. According to our intuitions described above, one must define the terms "infrequent", "explain",

and "make sense" (or the constant $\text{freq}_{max}$, and the relations $\vdash$ and $\succ$) to be able to define some algorithm to detect anomalies. In this research:

- $\text{freq}_{max}$ is arbitrarily defined to be 2% (and also 5%)
- the relation $\vdash$ is actually the relation "was mined from" that relates models with logs, under different algorithms for process mining
- we make no specific contribution to propose a particular definition of $\succ$, but instead we use whatever definition that was implicitly or explicitly defined in the mining algorithm.

*1.4. Objective and outline*

This paper proposes and compares three algorithms for anomaly detection of process executions. The first algorithm is the **threshold algorithm**. The algorithm is called threshold because it will consider as anomalous all traces that has a conformance with the mined model below a certain threshold. The second algorithm is the **iterative algorithm**, which is an extension of the threshold algorithm into an iterative procedure. At each iteration, only the trace with lowest conformance is considered an anomaly, and removed from the mining process in the next iteration. The third algorithm is the **sampling algorithm**. The algorithm is based on the following materialization of the basic intuitions: since anomalies are infrequent, a random sample of the trace-instances from the log will likely not include an anomaly. Then, models mined from this sample will probably not include the anomalies as instances,and traces that are not instances of this model can be considered anomalies.

These algorithms have many parameters that must be adjusted. The threshold algorithm must decide on a particular process mining algorithm, the metric to measure the conformance of a trace and a model, and the threshold value itself. The iterative algorithm also must define the mining algorithm, the conformance metric, and a stopping condition of iteration. Finally, the sampling algorithm must choose not only the mining algorithm but also the proportion of the traces that will be sampled.

In this paper, the definition of these parameters was achieved by a grid search on their values, using an artificially generated set of traces (for which we know which ones are the anomalies), and the final comparison of the algorithms was performed on a different but also artificially generated set of examples.

This paper is organized as follows: section 2 presents the algorithms for detecting anomalies; section 3 presents the algorithm to generate the artificial logs, and the evaluation of the algorithm on these logs. Section 4 discusses the related literature and compare, when possible, our approach to others. Section 5 discusses the limitation of this work.

## 2. Anomaly Detection Algorithms

*2.1. Threshold Algorithm*

The central idea of the Threshold algorithm is to explore the implicit relation $\succ$ defined by noisy process mining algorithms. Remember that a noisy miner

will prefer to build a "better" model (under $\succ$) even at the cost that some of the traces in the log will have a low conformance to the mined model. Thus, it is possible that the traces that the algorithm considered noise are exactly the anomalous ones.

Algorithm 1 presents the Threshold algorithm. There is only one change regarding the basic intuition described above. If a trace is a candidate for anomaly (that is, if the trace has frequency in the log of less than 2%), then the trace is removed from the traces given to the process mining (line 7); , and thus, it is more likely that a really anomalous trace will have very little conformance with the mined model.

**Algorithm 1:** Threshold Algorithm

**Input**: A log $L$
**Output**: A set of anomalous traces $T^A$.
**Parameter**: A threshold conformance value: $x \in (0, 1)$.
**Parameter**: A process discovery algorithm: `mine`.
**Parameter**: A conformance assessment algorithm: `conformance`.

```
1  T =the set of all classes of traces from the log L;
2  T^C = {} used to contain the anomalous candidate traces;
3  foreach t ∈ T do
4      if freq_L(t) ≤ 2% then
5          T^C = T^C + {t};
6  foreach t ∈ T^C do
7      T' = T − {t}; M = mine(T');
8      if conformance(M,t) < x then
9          T^A = T^A + {t};
10     return T^A;
```

The Threshold receives three parameters, the mine function, a noisy process discovering algorithm, the conformance function, that computes the conformance of a trace to a model, and the threshold value $x$. These parameters are not really inputs of the algorithm, since a user will have no way of deciding on them given a particular log. It is part of this research to show which choices of mine, conformance and $x$ are the best ones.

### *2.2. Iterative Algorithm*

The iterative algorithm (Algorithm 2) follows the ideas of the threshold algorithm, but instead of selecting all traces with conformance below the threshold, it will select only the trace with lowest conformance as an anomaly, and repeat the process, until the minimum conformance trace is above a threshold. The idea is that the threshold algorithm must detect all anomalies in a single step (by selecting all traces that are below the threshold), and that may be too tax-

ing for the mining algorithm. The iterative algoritms selects one trace at a time as an anomaly, the trace with the lowest conformance.

**Algorithm 2:** Iterative Algorithm

**Input**: A log $L$
**Output**: A set of anomalous traces $T^A$.
**Parameter**: A value $x \in (0, 1)$ for conformance threshold.
**Parameter**: A process discovery algorithm: `mine`.
**Parameter**: A conformance assessment algorithm: `conformance`.

```
1  T ←the set of all classes of traces from the log L;
2  T^C ← {} ;
3  foreach t ∈ T do
4      if freq_L(t) ≤ 2% then
5          T^C ← T^C + {t};
6  repeat
7      C_min ← 1;
8      foreach t ∈ T^C do
9          T' ← T − {t};
10         M ← mine(T');
11         cost ← conformance(M,t);
12         if cost < C_min then
13             C_min ← cost;
14             t_min ← t;
15     if C_min < x then
16         T^A ← T^A + {t_min};
17         T^C ← T^C − {t_min};
18 until C_min ≥ x;
```

*2.3. Sampling algorithm*

The sampling algorithm is based on the idea that a sample of the log should not contain an anomaly, since they are infrequent. Thus if one selects a sample of the log, mine the model from it, anomalies should not be instances of the model. Thus all infrequent that are not instances of the mined model are considered an anomaly. For each candidate anomalous trace (Line 3), the algorithm is executed as follows: (i) get a sampling of traces from the log (Line 7); (ii) create a process model for this sampling (Line 8); and (iii) test if the candidate anomalous trace is an instance of the discovered model (Line 9). Anomalous traces are those traces that are not instance of the process model discovered in each iteration.

**Algorithm 3:** Sampling Algorithm

**Input**: A log $L$
**Output**: A set of anomalous traces $T^A$.
**Parameter**: A sampling size: $s \in (0, 1)$
**Parameter**: A process discovery algorithm: `mine`.

```
1  T ←the set of all classes of traces from the log L;
2  T^C ← {} used to contain the anomalous candidate traces;
3  foreach t ∈ T do
4      if freq_L(t) ≤ 2% then
5          T^C ← T^C + {t};
6  foreach t ∈ T^C do
7      S ← sample of s% of traces of L;
8      M ← mine(S);
9      if t is not instance of M then
10         T^A ← T^A + {t};
```

*2.4. Implementation of the algorithms*

All algorithms presented in this paper are integrated with a set of process mining tools available in the ProM Framework. The ProM framework is a platform independent, open-source, "pluggable" environment for process mining [40]. The framework is flexible with respect to the input and output format, and it also allow for the easy reuse of code during the implementation of new process mining techniques.

All algorithms discussed above are implemented as scripts that make calls to the appropriate ProM functions. Both the mining algorithms and the conformance functions used in the algorithms are the ones available in the ProM framework.

## 3. Evaluation of the algorithms

We evaluated the three algorithms with a set of 360 synthetic logs, which have been created based on the traces of known process models randomly created. Two reasons influenced our choice for using synthetic data for assessment. First, there is no standard benchmark for anomaly detection of processes, and second, as [26] reported, even using logs from real processes, there are problems in labeling a trace as anomalous in real life situations. Therefore, to measure the efficacy of proposed algorithms one has to deal with artificial logs, where it is known which are the anomalous traces.

The evaluation carried out in this work can be summarized in three steps, as follows: (i) the creation of the 360 random logs described in Section 3.1; (ii) the use of a small random subset of the logs to determine the best values for the parameters of the three algorithms (see Section 3.2); (iii) the use of the

remaining logs in the evaluation of the three algorithms together with the naive algorithm (in Section 3.3).

*3.1. Method for the creation of the logs*

The experiments were based on 360 logs with different configurations. First, we created 60 random process models. These models were created by randomly combining OR-blocks, AND-blocks, LOOP-blocks and sequences (with a higher probability of choosing sequences). The models were well balanced, that is, blocks are always within other blocks. The models were created with 20, 35 and 50 components - a component is an activity, or an AND-, OR-, or a LOOP-split. We verified that all random models generated at least 10 different traces. The algorithm for the model creation is shown in Appendix A.

The 60 models generated by the algorithm had on average 940 different traces (minimum of 10 traces, by construction, and a maximum of 28665 traces). The models had an average of 17.7 different activities per model (minimum of 9, maximum of 29). Finally, the average trace length was 21.19 activities (minimum trace size was 1 activity, and maximum trace size was 78).

For each model, we generated a "normal" log as follows. We generated all traces of the log, with all loops repeated at most 2 times. We ordered these traces in a random order. For the first one, we selected a random number from 0 to 1000, with uniform probability. Let us call that number $n_1$, which is the multiplicity of the first trace in the log. The second trace would have a multiplicity $n_2$ randomly selected (with uniform probability) from 1000-$n_1$ to 1000. The third trace would have multiplicity randomly selected from the interval 1000-$(n_1 + n_2)$ to 1000. And so forth until the interval reduces to 0.

This procedure for generating a log will, on average, produce traces with an exponential distribution of frequencies. The expected multiplicity of the first trace is 500, for the second 250, for the third 125, and so on. The expected number of different traces in the log is 10 (and that is why we require each log to generate at least 10 different traces).

To generate an anomaly we randomly selected one of two alternatives:

- insert at a random place in the trace, an activity that is already present in the trace, or
- remove a random activity from the trace.

We verified that the anomaly generated by each of these two procedures was not an instance of the original model.

Each model generated 6 different logs with different characteristics or *profiles* regarding the anomalies. The anomaly profile is a combination of how many different anomalous traces are in the log (one or two), and how many times each anomalous trace is repeated in the log (once, three, or five times). The 6 logs generated from each model had the following profiles:

- one log had a single anomaly
- one log had a single anomaly repeated three times

- one log had a single anomaly repeated five times
- one log had two different anomalies, each occurring only once in the log
- one log had two different anomalies, each occurring three times in the log
- one log had two different anomalies, each occurring five times in the log

The approach used to create the anomalous traces represents our intuitions that the anomalous traces are quite similar to normal traces. Our anomalous traces are based on the same set of tasks but with "small changes" in the order or number of times a particular activity is executed. Furthermore, our logs incorporate the possibility that a particular fraud was repeated, and that more than one fraud is present. Of course, these logs do not exhaust the possibilities of frauds in process aware systems, but we believe that these logs cover the "difficult to detect" frauds.

*3.2. Parameterization of Algorithms*

All the three algorithms discussed in Section 2 have parameters that need determining. To use the threshold algorithm one need to define the mining algorithm, the conformance metric, and the threshold value. As discussed, both the mining algorithms and the conformance metrics were the ones available in the ProM framework. We tested the *alpha*, *alpha++*, *multiphase*, and the *heuristic* mining algorithms. For conformance metrics we used *fitness*, *behavioral*, *structural*, *appropriateness*, *size* metrics. For the threshold we used the values 0.5, 0.7, and 0.9. The iterative algorithm has the same parameters, and the same choices were used. Finally the sampling algorithm has the mining algorithm as choice (and we tested the same four choices above), and the sample ratio. We tested the sample ratios of 20%, 50%, and 70%.

We randomly selected 60 of the 360 artificial logs to determine these parameters. For each detection algorithm we selected all combination of parameters possible and measured the quality of the predictions regarding which were the anomalous traces in each of the 60 *training* logs.

In order to discuss the quality metric, let us briefly review the standard metrics for a binary classifier. The classes of a binary classifier are usually referred to as "positive" and "negative". In our case, "positive" refers to being an anomaly, and "negative" to being a normal trace. Furthermore:

- a true positive (TP) is an example (a trace in our context) that is classified as positive by the system and it is indeed positive.
- a false positive (FP) is an example that is classified as positive by the system, but it is really a negative example
- a true negative (TN) is an example that is correctly classified as negative by the system
- a false negative (FN) is an example that is classified as negative by the system, but it is really a positive example

The standard metrics for quality of a binary classifier are:

- accuracy = $(TP + TN)/(TP + TN + FP + FN)$
- precision = $TP/(TP + FP)$
- recall = $TP/(TP + FN)$

In the case of this research, because of the emphasis on anomalies as frauds, as discussed, we want to the false negatives to be as small as possible, which implies that recall will be as close to 1 as possible. But using only recall as the metric of quality of the algorithms is inappropriate, since by definition, the naive algorithm will have recall equal to one. In this case, the naive algorithm will have the highest recall, but in turn it will result on the lowest precision - since all infrequent traces will be marked as positive. The f-measure family of metrics tries to balance (with different emphasis) both precision and recall [41]. They are defined as:

- f-measure = f1-measure = 2.precision.recall/(precision + recall)
- f$\beta$-measure = $(1 + \beta^2)$.precision.recall/($\beta^2$precision + recall)

F1-measure is the harmonic mean between precision and recall. The f$\beta$-measure weights the recall by $\beta$ when performing the harmonic mean. For $\beta > 1$, recall will be increasingly more important when performing the harmonic mean, the larger t $\beta$.

As we discussed above, under the fraud perspective, recall should be as high as possible, but precision should also be high. Thus we need a metric that places more emphasis on recall, and we arbitrarily opted for the f4-measure. But we will also report the f1-measure, accuracy, and recall. The accuracy calculations involve the true negative (TN) which will always be very high, so the figures for accuracy will be deceptively high. So we only report the accuracy for the candidate traces, that is, the traces with frequency lower than 2%. We also consider the f1- and f4-measures to be 0 when either precision or recall are 0.

The resulting best parametrization of the three algorithms are:

**Threshold** Quality of the solution with the best parameters: f4-measure=0.481, f1-measure = 0.319, recall = 0.533, and accuracy = 0.435.

- Process discovery algorithm: alpha.
- Process model evaluation algorithm: fitness.
- Threshold factor: 0.9.

**Iterative** Quality of the solution with the best parameters: f4-measure=0.204, f1-measure = 0.158, recall = 0.221, and accuracy = 0.522.

- Process discovery algorithm: alpha.
- Process model evaluation algorithm: appropriateness.

- Threshold factor: 0.9.

**Sampling** Quality of the solution with the best parameters: f4-measure=0.913, f1-measure = 0.615, recall = 1, and accuracy = 0.501.

- Process discovery algorithm: heuristic miner.
- Sampling factor: 70%.

### *3.3. Execution and Results*

The three algorithms, with the parameters defined in the previous section, were executed in the remaining 300 logs. The results are in Table 1. The statistical analysis of the results (One-way ANOVA followed by Tukey HSD [33]) show that the difference in f4-measure for the sampling and the naive algorithm is not significantly different (p-values = 0.784) and all other differences are. Thus, in terms of f4-measure, the sampling algorithm is equivalent to the naive, followed by the threshold, and finally the iterative. In terms of f1-measure, the order is sampling, naive, threshold, and finally iterative (all differences are significative). In terms of accuracy (for the candidate traces set), the order is iterative, sampling, threshold, and finally naive (the difference between threshold and sampling is not significative, all other are). The table also report the average execution time per log.

Table 1: Performance for execution of the algorithms on the test group.

| | Threshold | Iterative | Sampling | Naive |
|---|---|---|---|---|
| F4-measure | 0.305 | 0.131 | *0.886* | 0.867 |
| F1-measure | 0.181 | 0.084 | *0.561* | 0.476 |
| Recall | 0.345 | 0.147 | 0.993 | 1.000 |
| Accuracy | 0.432 | *0.578* | 0.467 | 0.331 |
| Time (s) | 22.5 | 51.9 | 28.6 | 0 |

We also carried out an analysis to verify if there were a performance difference among the algorithms when executed on logs of different profiles. We tested each algorithm only on (i) logs with one or two classes of anomalous traces and (ii) logs in which the same class of anomalous traces occurred once, three times, or five times. The results are not reported in this paper, but in each case the order of the quality of the algorithms (regarding the f4-measure) was the same, that is: sampling and naive are not statistically different, followed by threshold, and iterative.

We also tested the sensibility of our results to an early decision, that is, that the cutoff frequency for anomalies was 2%. We ran the sampling and the naive algorithms with the cutoff frequency set to 5% on the same 300 test logs. Notice that the true anomalies frequencies in the logs were the same, but now the algorithms would consider a much larger set of candidates (all traces with frequency at most 5%). The average f4-measure for the sampling algorithm was 0.871, while for the naive was 0.841. The difference is significative (paired t-test,

p-value=4.54e-9). That is, the f4-measure for both algorithms reduced a little from the results with 2% cutoff, but now the sampling algorithm is significantly better than the naive. The same results are true for the f1-measure.

Finally we experimented and compared the sampling algorithm (with the optimal parametrizations above) and the naive algorithm for the 300 test logs *without* anomalies The average number of (false) positives for the naive algorithm was 3.60 traces, while for the sampling the average was 2.82. The difference is significative (paired t-test, p-value<2e-16). Thus, for logs in which there are no anomalies, the sampling algorithm returns significantly less false negatives than the naive algorithm, and thus reduces the cost of the necessary human analysis of those traces.

### *3.4. Further experiments*

Before the experiments described above, we run the algorithms with another set of logs. These logs were also randomly generated from a set of random models, but these models were much simpler than the ones discussed above. In particular, these models were "smaller", that is, they contained less activities (at most 20 different activities in them), and they did not contain loops. Furthermore the process of generating the log, and in particular the multiplicity of each trace was different. For those models we associated a probability for each branch of an OR-split (in the same way as the stochastic information control nets proposed in [17]) and generated the traces based on those probabilities of choosing one or the other branch of each OR-split.

For these logs, we followed the training/test protocol for the previous experiment. But since the modes were simpler, we generated many more logs, 300 for the training, and 1500 for the test procedure.

The first important result in this experiment is that the best parametrization of the algorithms were the same as for the experiments described in 3.2. This provide some evidence that the parametrization of the algorithm is stable across very different set of logs. This is important to a potential user of the algorithms since one will likely have very few logs, none of which will have known anomalies. Thus the user will not be able to use a separate set of logs to tune the algorithms, and must rely on the parametrization discussed in this paper.

The second important result is that although all algoritms performed better with that logs (higher figures for the f4 and f1 measures) the relative order of the algorithm remained the same, with the sampling algorithm outperforming the naive, which outperformed the threshold, which in turn was better than the iterative algorithm. Furthermore, in this case, the difference between the sampling algorithm and the naive was statistically significant.

This result show that although the sampling algorithm had a similar result to the naive in the previous experiment regarding the f4-measure, it is likely that in general the sampling outperforms the naive. Besides the difference in performance regarding the f4-measure for this experiment, and the one with 5% cutoff, one should also consider that the sampling algorithm has better results if the costs of a false negative and a false positive are more balanced, as measured

by the f1-measure, for example. If a false positive cannot be accepted, which would correspond to a f$\infty$-measure, then by design the naive algorithm is the best. At the balance represented by the f4, where recall is four times more important than precision, the sampling seems slightly better than the naive, sometimes with statistically equivalent results, and sometimes with significantly better results.

### *3.5. Further analysis of the sampling algorithm*

Once we determined that the sampling algorithm is the best detector, we carried out some further analysis to discover the dependency of the algorithm performance on the different characteristics of the log such as: number of anomalous classes (one or two), number of anomalous occurrences (one, three, or five), and number of candidate traces (traces with frequency at most 2%).

**Number of classes of anomalies** This characteristic influences the performance of sampling algorithm. The f4-measure for logs with only one anomaly class is 0.859, and for logs with two anomalies is 0.916. The difference is significative (t-test, p-value=3.4e-07). Thus the sampling algorithm performs better on logs with more classes of anomalous traces. The same is true for the f1-measure.

**Number of anomalous trace occurrences** This characteristic does not influence the performance of algorithm. The f4-measure for logs with only a single instance of each anomalous traces is 0.878, for three instances is 0.897 and for five instances 0.883. The difference is not significant (ANOVA test p-value=0.77). The same is true for the f1-measure.

**Number of candidate traces** This is a characteristic that negatively influences the performance of sampling algorithm, the larger the set of candidate traces, the lower the performance of the algorithm. A linear regression of the f4-measure and the size of the set of candidate traces yields a multiplicative coefficient of -0.049, and although this is a small value, it is significantly different than 0 (p-value = 3.5e-13).

### *3.6. Example with a real log*

As an example of running the Sampling and Naive algorithm in a real log, we obtained the log used in [8] to illustrate another process anomaly detection. The log is derived from the different information systems used by a Dutch municipality to support handicapped and mobile restricted citizens. The log include all complete traces that started and ended within the January 2007 to August 2008 period, and it contain 796 traces. The activities in the log may be grouped into 10 different activities; the longest trace has 12 instances of activities, the shortest has 5. On average the traces have 6 activities.

The method proposed in [8] used a series of "filtering" and mining steps. The idea of the method is finding the "most appropriate model" that is mined from the log, and flagging as anomalous the traces with low conformance to

that model. But in the process of finding the most appropriate model one may remove all instances of an activity from all the traces in the log if the inclusion of that activity reduces significantly the appropriateness of the mined model. That is, the method loops through the steps of mining the most appropriate model (in that case using heuristic mining) and removing all instances of one or more activities that makes the mined model particularly complex. [8] does not propose an automatic method to select the activities that are "causing" the complexity of the mined model, and thus that selection was necessarily performed by a specialist. Once the activity or activities are selected, all instances of those activities are removed from all traces. The process will then repeat the mining step, followed by further selection, until the resulting model has a high enough appropriateness (in that case 80%).

The result of the most appropriate model method described in [8] to the Dutch municipality logs required the filtering out of two activities, and resulted in 6 traces as potential anomalies. Notice that we have no independent confirmation that those 6 traces were anomalous, or that there was no other anomalies among the traces deemed "normal". Thus by using this log we are not providing an independent measure of the accuracy (or other quality metrics) of the Sampling algorithm; we are showing that the sampling algorithm can be applied to a real log (one whose generation process may or may not be compatible with the one described in 3.1), and compare its results with a different algorithm for anomaly detection that is very costly since it demands human intervention in the scoping process.

The Sampling algorithm generated 15 traces as potential anomalies, among them all the 6 defined by the most appropriate model algorithm. The Naive approach generated the same 15 examples plus a new one. This shows that the Sampling algorithm returned a reasonable set of anomaly candidates when compared with a different and more costly method.

## 4. Related Work

Data mining community has published a large and growing body of literature in anomaly detection. To cite a few that centers on the idea of fraud detection, [15] presents how data mining techniques can be used to early detect inside information in option trading. In [18] the authors present a system which is used to detect fraudulent usage of a cellular phone (cellular cloning). There are solutions concerned with the intrusion detection in networks [23, 25]. Other efforts are concerned with the detection of frauds in auctions or e-commerce sites [26]. None of these works deal explicitly with processes and traces, they refer only to the data values of the cases.

### *4.1. Process mining*

Process mining is a method used to reconstruct a process model from a log generated by a system. Such a technology is an alternative to construct models from scratch, and in the last twelve years it has raised the attention

of researchers around the world [37, 12, 39]. For this work, process mining algorithms are important to help us find a model that will be used as a classifier of traces as anomalous or normal.

The field of process mining was first coined in the context of software processes. Cook and Wolf [11] present a process discovery as a tool to support the design of software processes. Also a precursor work in process mining, the paper by Agrawal et al [3] present an algorithm that mine finite state models with three properties: completeness, minimality, and no redundancy. Complete in the sense that a process model may contain all task and dependencies between them that there is in the log; minimal in the sense that the mined process has the minimum set of structural elements; and non redundancy in the sense that the process model does not play an instance by different ways. The properties of the mined process reported in this work are the first support to the "make more sense" relation among two processes models.

Many process mining algorithms have been proposed in the last years [37, 39, 38, 31, 22, 47, 21, 14, 43]. Of relevance to this paper is the distinction between precise and noisy mining algorithms. The precise algorithms will generate a model that includes all traces in the log. The precise mining algorithms are usually based on on rewriting techniques [31, 43], but considering that some constraints in the log are satisfied, one may also consider the $\alpha$-algorithm [39] as a precise algorithm.

Noisy approaches will consider that some of the traces in the log are noise and need not to be included in the mined model. Noisy mining algorithms [3, 47, 10, 27, 22, 21, 45, 46] in their majority use the frequency of the temporal order relation between two activities to infer their dependency, and thus infrequent dependencies in the log may not be modeled in the resultant process model. An exception to the frequency based approach is the work on genetic mining [14, 13] which uses a search based on genetic algorithms with an explicit representation of the ordering among models.

As we mentioned, there has been some recent work on structural metrics on process models [29, 42] and on the relation between models and logs[44, 20, 19]. These works use the concepts of: (i) fitness[30, 28], which assess the portion size of the log that can be correctly played by the model, that is, the degree of completeness; (ii) behavioral appropriateness[28, 34], which measures the degree of predictability to support the execution of unseen trace in the log; (iii) structural appropriateness[28, 34], which assess the complexity of the process model; and (iv) appropriateness[8], which was suggested to be a balance between structural and behavioral appropriateness.

### *4.2. Anomaly Detection in Process Logs*

There is a limited published work in detecting anomalies in process logs, besides our own [5, 4, 6, 8, 7]. Aalst and Medeiros [36] present two anomaly detection methods that are supported by $\alpha$-algorithm. A drawback of this work is that it demands a known "normal" log, which is used to mine a process model that will classify the traces of an audit log. Instances with low conformance are the anomalous traces. However, a known "normal" log may not be available in

applications domains that demands a high flexible support because the normal instances change constantly.

Yang and Hwang [48] present a framework for automatic construction of a model for detecting health care fraud and abuse. In that work clinical pathways are used to construct a detection model, whose features are based on frequent control-flow patterns inferred from two datasets, one with fraudulent instances and other with normal instances. Similarly to approach reported in [36], data set of normal and fraudulent instances may not be available in some applications, which is clearly a serious limitation.

Closely related to the algorithms reported here are the algorithms presented in [5, 4, 6]. However, because these algorithms are based on incremental mining [43], they have severe scalability limitations, both regarding the number of activities in the traces, and the number of different traces in the log. Also, that work use the measure of number of changes that are necessary to include a trace into a model to identify anomalies (forcing an anomaly into the model will cause many changes to the model itself) while this work considers the difference of conformance between model and log.

Accorsi et al [1, 2] present an approach for detecting illegal data flow execution in business processes. These works propose a forensic analysis tool called RecIF, that constructs a graph of data flow among subjects (propagation graph) in a process. Then, a recorded set of polices are verified regarded compliance against the propagation graph constructed from the log. The output of processing are a set of evidences of frauds or deviations from polices.

Although the work in [1, 2] have similarities with our anomaly detection approaches (e.g. model construction and conformance test), there exist one important differences. Our work does not apply a direct conformance test method, because in our case there is no predefined polices or rules to test the conformance. The central aspect of our work is that there is no a priori known correct model.

## 5. Conclusions and discussion

Detection of anomalies are a growing research area. It has been realized that fraud and intrusion detection in systems, novelty detection in time series, bio-threats discovery, and other similar tasks are better understood as anomaly detection (see [9] for a list of current literature in each of these applications). Of course, fraud or intentional threats are not the only source of anomalies, but this has been the perspective adopted in this research.

This work presented three new algorithms (threshold, iterative, and sampling) to detect "hard to find" anomalies in a process log based only on the control-flow perspective of the traces. This work does not deal with anomalous executions of processes that follow a correct execution path but deal with unusual data, or are executed by unusual roles or users, or have unusual timings. We center our approach on the control-flow perspective alone. The anomalies are hard to find because they are similar to the normal traces themselves - there

is no unique characteristic in the anomalous traces that make them different from the normal logs except that they are not instances of the "correct" model that generated the normal traces. The anomalies are hard to find also because under the fraud perspective, they were intentionally made difficult to detect.

The algorithms proposed in this research are based on the idea that anomalies are infrequent, although in our examples some normal traces are also infrequent. In this research infrequent is arbitrarily chosen to the 2%. But, as we discussed in section 3.3, the results do not change much if we use a 5% cutoff.

Among the three algorithms, the sampling algorithm with 70% sampling rate and heuristic mining was shown to be the one with better f4-measure (and f1-measure) to detect these anomalies. The result was achieved by testing the three algorithms together with a naive algorithm, that only selects as anomalies the infrequent traces, in 300 artificially generated logs (of a 1000 traces each) for which it was known which were the anomalies. The iterative algorithm is the best for the standard measure of accuracy.

The sampling algorithm was also applied to a real log, for which we had no independent classification of which trace was anomalous, but the algorithm detected all the anomalies detected by a different method that require user choices. Thus, the sampling algorithm can substitute a costly procedure to detect anomalies.

There are two clear limits to this research. The first one is that we only adopt the control-flow perspective, and thus only traces that follow a "different path" from normal traces are detected. In real life, anomalies can be limited not only to the control-flow perspective, but also to the data, and organizational perspectives. But, the anomalies may also be due to the combination of these perspectives: a particular execution path may be used only when executed by senior roles, so a particular trace that followed that path is not in itself anomalous, neither are traces whose activities were executed by junior roles. But a trace that followed that path and had activities executed by a junior role are anomalous, and this determination involved both the control-flow and the organizational flow perspectives taken together. Other examples dealing with the other perspectives can also be constructed. Thus, the approach taken by this research is incomplete and should be understood as a stepping stone towards more complete methods. Even problems where the combination of perspectives is needed can benefit from a single perspective approach as described in this research - a simple and possible solution to problems that involve the combination of perspectives is to combine simple anomaly detectors: ours for control flow data, and some others for say the data perspective, so that if any detector points out a trace as anomalous, it should be flagged as anomalous. This is a high recall, low precision approach that may be appropriate for some problems.

The second limit of this research is a more technical one. How far should one generalize the main result of this paper - that the sampling algorithm achieves the best results in detecting the anomalies as measured in different metrics (f4-measure, f1-measure, and accuracy) - to other logs? The only strong claim we can make is that since out test logs were a random sample of logs from a population of logs created with the method described in section 3.1, and since

the sampling algorithm performed significantly better than the alternatives for that sample, then one can conclude (with 95% confidence) that it will perform better in any other subset of that population. Any other conclusion is a leap of faith, which depends on the reader's belief on how *representative* are our artificial logs regarding "real" logs. If the logs used here are representative of future logs, one can be more sure that the conclusions discovered in this paper will hold for these other logs.

There are at least two components to determining how representative the logs are: how realistic are the models and how realistic are the distribution of the multiplicities of traces in each log. Unfortunately, we have no other source than our own intuitions regarding both. We described in details the model generation algorithm (section Appendix A) and the statistics of the generated models (section 3.1) so that the reader can evaluate how similar our models are to his or her own problems. For example, our traces were on average 21 activities long. If the one is planning of using our results in logs with traces of hundreds of activity long, one should be less confident that the results will hold.

The second component is the distribution of the multiplicity of the traces. Our distribution is very unequal, with an expected exponential distribution. If one's distribution is more uniform, with all traces having similar frequencies, than one should be less confident that the results will hold. In particular, in the case of uniform distributions, the naive algorithm should have a much better performance.

Finally, the models and the logs used in this paper are available online[3], so that researchers can propose new algorithms and fairly compare them to ours, and future users of our algorithms may evaluate if our models and logs are representative of the models and logs they will face in their problems.

### *5.1. Acknowledgments*

We would like to thank Prof. Wil van der Aalst and his group for fruitful discussions regarding this research line, and for the access to the Dutch municipality log.

## References

[1] Accorsi, R., Wonnemann, C., 2010. Auditing workflow executions against dataflow policies. In: Business Information Systems. Vol. 47 of Lecture Notes in Business Information Processing. Springer, pp. 207–217.

[2] Accorsi, R., Wonnemann, C., Stocker, T., 2011. Towards forensic data flow analysis of business process logs. In: Sixth International Conference on IT Security Incident Management and IT Forensics (IMF). IEEE Press, pp. 3–20.

[3] `http://www.fabiobezerra.pro.br/logs/`

[3] Agrawal, R., Gunopulos, D., Leymann, F., 1998. Mining process models from workflow logs. In: EDBT '98: Proceedings of the 6th International Conference on Extending Database Technology. Vol. 1377 of LNCS. Springer, pp. 469–483.

[4] Bezerra, F., Wainer, J., 2008. Anomaly detection algorithms in business process logs. In: 10th International Conference on Enterprise Information Systems. SciTePress, pp. 11–18.

[5] Bezerra, F., Wainer, J., 2008. Anomaly detection algorithms in logs of process aware systems. In: SAC '08: Proceedings of the ACM Symposium on Applied Computing. pp. 951–952.

[6] Bezerra, F., Wainer, J., 2008. Auditing workflow logs for fraud detection. In: Proceedings of the KDD 2008 Workshop on Data Mining for Business Applications. pp. 1–5, `http://labs.accenture.com/kdd2008_workshop/dmba_proceedings.pdf` accessed 12/2011.

[7] Bezerra, F., Wainer, J., 2011. Fraud detection in process aware systems. International Journal of Business Process Integration and Management (IJBPIM) 5 (2), 121–129.

[8] Bezerra, F., Wainer, J., Aalst, W. M. P., 2009. Anomaly detection using process mining. In: Enterprise, Business-Process and Information Systems Modeling. Vol. 29 of Lecture Notes in Business Information Processing. Springer, pp. 149–161.

[9] Chandola, V., Banerjee, A., Kumar, V., 2009. Anomaly detection: A survey. ACM Comput. Surv. 41 (3), 1–58.

[10] Cook, J. E., Du, Z., Liu, C., Wolf, A. L., 2004. Discovering models of behavior for concurrent workflows. Computers in Industry 53 (3), 297–319.

[11] Cook, J. E., Wolf, A. L., 1998. Discovering models of software processes from event-based data. ACM Trans. Softw. Eng. Methodol. 7 (3), 215–249.

[12] de Medeiros, A., van der Aalst, W., Weijters, A., 2003. Workflow mining: Current status and future directions. In: On The Move to Meaningful Internet Systems: CoopIS, DOA, and ODBASE. Vol. 2888 of LNCS. Springer, pp. 389–406.

[13] de Medeiros, A. K. A., 2006. Genetic process mining. Ph.D. thesis, Technische Universiteit Eindhoven, Eindhoven, The Netherlands.

[14] de Medeiros, A. K. A., Weijters, A. J. M. M., van der Aalst, W. M. P., September 2006. Genetic process mining: A basic approach and its challenges. In: Business Process Management Workshops. Vol. 3812 of LNCS. Springer, pp. 203–215.

[15] Donoho, S., 2004. Early detection of insider trading in option markets. In: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. KDD '04. ACM Press, pp. 420–429.

[16] Dumas, M., van der Aalst, W., ter Hofstede, A., 2005. Process-Aware Information Systems: Bridging People and Software through Process Technology. Wiley, ISBN 13 978-0-471-66306-5.

[17] Ellis, C., Kim, K., Rembert, A., Wainer, J., 2011. Investigations on stochastic information control nets. Information SciencesIn press. avaliable in http://dx.doi.org/10.1016/j.ins.2011.07.031.

[18] Fawcett, T., Provost, F., January 1997. Adaptive fraud detection. Data Mining and Knowledge Discovery 1, 291–316.

[19] Goedertier, S., Martens, D., Vanthienen, J., Baesens, B., 2009. Robust process discovery with artificial negative events. Journal of Machine Learning Research 10, 1305–1340.

[20] Greco, G., Guzzo, A., Ponieri, L., Sacca, D., 2006. Discovering expressive process models by clustering log traces. IEEE Transactions on Knowledge and Data Engineering 18 (8), 1010–1027.

[21] Hammori, M., Herbst, J., Kleiner, N., 2006. Interactive workflow mining - requirements, concepts and implementation. Data & Knowledge Engineering 56 (1), 41–63.

[22] Herbst, J., Karagiannis, D., 2004. Workflow mining with inwolve. Computers in Industry 53 (3), 245–264.

[23] Lee, W., Xiang, D., 2001. Information-theoretic measures for anomaly detection. In: IEEE Symposium on Security and Privacy. pp. 130–143.

[24] Neiger, D., Churilov, L., Flitman, A., Neiger, D., Churilov, L., Flitman, A., 2009. Introducing value-focused process engineering. In: Value-Focused Business Process Engineering : a Systems Approach. Vol. 19 of Integrated Series in Information Systems. Springer, pp. 1–13.

[25] Noble, C. C., Cook, D. J., 2003. Graph-based anomaly detection. In: KDD '03: Proceedings of the 9th ACM SIGKDD International Conference on Knowledge discovery and data mining. ACM Press, pp. 631–636.

[26] Pandit, S., Chau, D. H., Wang, S., Faloutsos, C., 2007. Netprobe: a fast and scalable system for fraud detection in online auction networks. In: WWW '07: Proceedings of the 16th international conference on World Wide Web. ACM Press, pp. 201–210.

[27] Pinter, S. S., Golani, M., 2004. Discovering workflow models from activities' lifespans. Computers in Industry 53 (3), 283–296.

[28] Rozinat, A., de Medeiros, A. A., Günther, C., Weijters, A., van der Aalst, W., 2007. Towards an evaluating framework for process mining algorithms. Tech. rep., Technische Universiteit Eindhoven.

[29] Rozinat, A., van der Aalst, W., March 2008. Conformance checking of processes based on monitoring real behavior. Information Systems 33 (1), 64–95.

[30] Rozinat, A., van der Aalst, W. M. P., 2005. Conformance testing: Measuring the fit and appropriateness of event logs and process models. In: Business Process Management Workshops. Vol. 3812 of LNCS. Springer, pp. 163–176.

[31] Schimm, G., 2004. Mining exact models of concurrent workflows. Computers in Industry 53 (3), 265–281.

[32] Shishkov, B., Sinderen, M., Verbraeck, A., 2009. Towards flexible inter-enterprise collaboration: A supply chain perspective. In: Enterprise Information Systems. Vol. 24 of Lecture Notes in Business Information Processing. Springer, pp. 513–527.

[33] Steinberg, W., 2010. Statistics Alive!, 2nd Edition. SAGE Publications, Ch. 26 - Tukey HSD Test, pp. 318–324.

[34] van Aalst, W. M., van Hee, K. M., van Werf, J. M., Verdonk, M., March 2010. Auditing 2.0: Using process mining to support tomorrow's auditor. Computer 43 (3), 90–93.

[35] van der Aalst, W. M. P., 2011. Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer.

[36] van der Aalst, W. M. P., de Medeiros, A. K. A., February 2005. Process mining and security: Detecting anomalous process executions and checking process conformance. Electronic Notes in Theoretical Computer Science 121 (4), 3–21.

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

[38] van der Aalst, W. M. P., Weijters, A. J. M. M., 2004. Process mining: a research agenda. Computers in Industry 53 (3), 231–244.

[39] van der Aalst, W. M. P., Weijters, T., Maruster, L., 2004. Workflow mining: Discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering 16 (9), 1128–1142.

[40] van Dongen, B., de Medeiros, A., Verbeek, H., Weijters, A., van der Aalst, W., 2005. The ProM Framework: A new era in process mining tool support. In: Applications and Theory of Petri Nets. Vol. 3536 of LNCS. Springer, pp. 444–454.

[41] van Rijsbergen, C. J., 1979. Information Retrieval. Butterworths.

[42] Vanderfeesten, I., Reijers, H., Mendling, J., van der Aalst, W., Cardoso, J., 2008. On a quest for good process models: The cross-connectivity metric. In: Advanced Information Systems Engineering. Vol. 5074 of LNCS. Springer, pp. 480–494.

[43] Wainer, J., Kim, K., Ellis, C. A., Setembro 2005. A workflow mining method through model rewriting. In: Groupware: Design, Implementation, and Use: 11th International Workshop. Vol. 3706 of LNCS. Springer, pp. 184–191.

[44] Weidlich, M., Polyvyanyy, A., Desai, N., Mendling, J., Weske, M., 2011. Process compliance analysis based on behavioural profiles. Information Systems In Press, Uncorrected Proof, 1009–1025.

[45] Weijters, A., van der Aalst, W., 2003. Rediscovering workflow models from event-based data using little thumb. Integrated Computer-Aided Engineering 10 (2), 151–162.

[46] Weijters, A., van der Aalst, W., de Medeiros;, A. A., 2006. Process mining with the heuristicsminer algorithm. Tech. rep., Beta Research School for Operations Management and Logistics.

[47] Wen, L., Wang, J., Sun, J., 2006. Detecting implicit dependencies between tasks from event logs. In: Frontiers of WWW Research and Development - APWeb 2006. Vol. 3841 of LNCS. Springer, pp. 591–603.

[48] Yang, W.-S., Hwang, S.-Y., July 2006. A process-mining framework for the detection of healthcare fraud and abuse. Expert Systems with Applications 31 (1), 56–68.

## Appendix A. Model generation algorithm

Since the models are well-balanced, that is, all blocks are within other blocks, we use a linear representation of models. `seq`(A,B). represents the sequence of a sub-workflow A followed by the sub-workflow B. `or`(A,B) represents an OR-split/join: the two branches are A and B. `and`(A,B) represents a AND-split/join of the two branches A, and B. Finally, `loop`(A,B) represent the loop that will either execute only A, or A followed by B and then by A (ABA), or ABABA, and so on.

The algorithm 4 is a stochastic algorithm that must make random choices at certain points. We use the notation **switch** *randomly choose* **do** to indicate a choice point in the algorithm, and **case** 40% and **otherwise** to indicate the probability of each choice. **case** 40% $A$ indicates that with probability 0.40 the choice $A$ must be followed. **otherwise** indicates the probability that is necessary so that all the probabilities of the choices of a **switch** add up to 1.

Finally, the auxiliary function `rndsplit`($n$) generates two integers $n_1$ and $n_2$, both positive, so that $n_1 + n_2 = n$.

**Algorithm 4:** Algorithm to generate models

**input** : $n$
**output**: A model of size $n$

```
1  if n==0 then
2      return ε;
3  else if n==1 then
4      return generate a new activity;
5  else
6      switch randomly choose do
7          case 60%
8              n1,n2 ← rndsplit (n − 1);
9              return seq(CreateModel (n1),CreateModel (n2));
10         otherwise
11             switch randomly choose do
12                 case 40% insert activity
13                     a ← generate new activity;
14                     return seq(a,CreateModel (n − 1);
15                 case 30% insert OR
16                     switch randomly choose do
17                         case 30% empty branch
18                             return or(CreateModel (n − 1),ε);
19                         otherwise
20                             n1,n2 ← rndsplit (n − 1);
21                             return or(CreateModel (n1),CreateModel
                               (n2));
22                 case 20% insert LOOP
23                     switch randomly choose do
24                         case 30% empty branch
25                             return loop(CreateModel (n − 1),ε);
26                         otherwise
27                             n1,n2 ← rndsplit (n − 1);
28                             return loop(CreateModel (n1),CreateModel
                               (n2));
29                 case 10% insert AND
30                     n1,n2 ← rndsplit (n − 1);
31                     return and(CreateModel (n1),CreateModel (n2));
```