UNIVERSITÄT AUGSBURG
Universitätsbibliothek

OPEN ACCESS
OPUS AUGSBURG

Taking on infrequent behavior in event logs using hypothesis tests

Patrizia Schalk, Lisa Petrak

Angaben zur Veröffentlichung / Publication details:

Schalk, Patrizia, and Lisa Petrak. 2022. "Taking on infrequent behavior in event logs using hypothesis tests." In *ATAED 2022: Algorithms & Theories for the Analysis of Event Data 2022 - Proceedings of the Workshop on Algorithms & Theories for the Analysis of Event Data co-located with the 43rd International Conference on Application and Theory of Petri Nets and Concurrency (Petri Nets 2022), Bergen, Norway, June 19-24, 2022*, edited by Robert Lorenz, Jan Martijn van der Werf, and Sebastiaan J. van Zelst, 25–42. Aachen: CEUR-WS.

Nutzungsbedingungen / Terms of use: **CC BY 4.0**

Dieses Dokument wird unter folgenden Bedingungen zur Verfügung gestellt: / This document is made available under these conditions:
CC-BY 4.0: Creative Commons: Namensnennung
Weitere Informationen finden Sie unter: / For more information see:
https://creativecommons.org/licenses/by/4.0/deed.de

Taking on Infrequent Behavior in Event Logs using Hypothesis Tests

Patrizia Schalk[1], Lisa Petrak[1]

[1]University of Augsburg, Universitätsstraße 6a, 86159 Augsburg

Abstract

When an event log is generated based on the real-life data of an existing process, there is a high probability that apart from events that happen frequently and therefore should be represented in the process model, infrequent behavior is featured in the event log that would make a model generated from this log hardly readable. There are many process discovery algorithms that work with such outliers by filtering infrequent behavior just before or during the creation of an appropriate process model. Less common methods make use of statistical tests to detect infrequent behavior in a preprocessing step. This paper aims to simplify the use of hypothesis tests introduced by Petrak et al. and proposes an approach to delete infrequent behavior without generating unwanted side effects for the process model. Furthermore, we solve a problem regarding the use of hypothesis tests for a process that contains loops and compare the results of hypothesis tests to well-known discovery techniques.

Keywords

Process Mining, Process Discovery, Filtering Outliers, Detection of Infrequent Behavior, Hypothesis Test, Directly Follows Graph, Loop Shortening

1. Introduction

Finding mathematical models for real-life processes is a highly relevant task, since such models can easily be analyzed in a way that shows exactly where the process has its weaknesses and where it can be optimized. However, constructing such a process model by hand leads to results that describe what the process should look like instead of how it actually looks. Therefore, it is highly desired to create such models automatically and analyze them afterwards, so there is no bias present in the model. To automatically construct such a model, a record of the behavior of the process, e.g. in the form of an event log, is needed. However, there are several challenges that make it hard to construct a model from an event log, especially if such a model has to meet some criteria like being readable.

One example of such a challenge is the existence of outliers in the event log – data that is present in the log but is so infrequent that it does not belong to the important parts of a process. Related to the existence of outliers is the existence of noise – data in the event log that cannot occur in the process. There are several causes for noise, such as human error ("I meant to log a different task than I actually did") or technical failure ("A part of the system shut down and didn't log a status during a period of time"). While noise is erroneous behavior and should not be featured in the event log, outliers can very well be part of the actual behavior of the process.

Algorithms & Theories for the Analysis of Event Data (ATAED) 2022

patrizia.schalk@informatik.uni-augsburg.de (P. Schalk); lisa.petrak@informatik.uni-augsburg.de (L. Petrak)

© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

CEUR-WS.org/Vol-3167/paper2.pdf

However, removing edge cases of the event log promises to make the remaining behavior and therefore a suitable process model more clear and readable. Finding such a reasonable model despite the existence of outliers in the event log is the task of process discovery. Most such techniques handle noise by using heuristics or by setting thresholds for the maximum amount of behavior that may be deleted [1, 2, 3, 4, 5, 6]. Fewer discovery techniques make use of statistical theory to detect infrequent behavior in event logs [7].

As mentioned before, a discovered model needs to meet some quality measures. For example, we want a process model that is able to replay most of the behavior present in the event log (but not everything, since infrequent behavior should be excluded) – this trait is measured by the *fitness* of a process model. However, fitness alone is not enough for a good process model, since we can accomplish perfect fitness by simply allowing every behavior, even behavior that has nothing to do with the process. Therefore, another quality measure is the *precision*, which ensures that the process model cannot replay way more than is present in the event log. The notion of *generalization* describes how open the model is for reasonable extensions, i.e. behavior that was not recorded in the log but can happen in the process. Finally, a process model should be simple and easy to read, so that a human can understand and analyze it – this trait is measured by the *simplicity* of the discovered process model. More information on quality measures for process models can be found in [8].

In this paper, we continue research on the question of how to discover a "good" process model for the main behavior of an event log by filtering out infrequent behavior. The idea to use hypothesis tests to find such outliers in an event log was presented in [7], where one-sided hypothesis tests are used to decide whether the direct neighborhood of two events should be classified as main or infrequent behavior. However, the approach of said paper creates unwanted side effects by deleting single events out of a trace and therefore creating new direct neighborhood relations that might be simply wrong. We present a new approach without this problem that is graphical, therefore easy to follow, and gives the user more power over the result he wishes to get. We first lay the foundation for our approach by defining needed notions in Section 2 and revisit the general idea to use hypothesis tests to find infrequent behavior in Section 3 by simplifying the used techniques described in [7] and investigating a running example that is small and easy to follow. In Section 4 we propose to use a Directly Follows Graph to represent the directly follows relations between events after we detected infrequent direct neighbors using hypothesis tests and show how we can delete infrequent behavior without risking side effects. We address a problem concerning loops that feature many iterations in the event log in Section 5 and propose a simple solution that is based on the well-studied Chinese Postman Problem [9]. Section 6 evaluates the results by comparing the accomplished fitness, precision, generalization and simplicity with those of well-known discovery techniques, namely the Inductive Miner infrequent [6] and the Directly-Follows Miner [5], with which our approach shares similar ideas. Section 7 concludes the paper.

2. Basic Definitions

We denote the set of natural numbers $\{1, 2, 3, \dots\}$ by $\mathbb{N}$ and define $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$. For an arbitrary set T we call $m : T \to \mathbb{N}_0$ a *multiset* over T. For any $a \in T$, $m(a)$ is the number of occurences of the element a in the multiset m. We write $a \in m$ if this number is greater than zero, i.e. $a \in m :\Leftrightarrow m(a) > 0$. We also write a finite multiset m over T with elements $a_1, \dots, a_n$ in the form $[a_1^{m(a_1)}, \dots, a_n^{m(a_n)}]$.

Definition 1 (Event, Trace, Event log [10]). *Let T be a set of activity names. A sequence of activities $\sigma = \langle t_1, \dots, t_n \rangle \in T^*$ is called a* trace. *An activity $a \in T$ is* contained in *σ if it occurs in σ at any time, i.e. $a \in \sigma :\Leftrightarrow \sigma = \langle t_1, \dots, t_n \rangle \wedge \exists i \in \{1, \dots, n\} : t_i = a$. For an arbitrary trace $\sigma = \langle t_1, \dots, t_n \rangle \in T^*$ with $n \geq 1$ we define $first(\sigma) := t_1$ and $last(\sigma) := t_n$.*
An event log *$L : T^* \to \mathbb{N}_0$ over T is a multiset of traces. The occurrence of an activitiy $a \in T$ in L is called an* event.

Regarding a real-life process, we give every event that takes place a name, which can be an artificial one (like $a, b, c, \dots$) or a descriptive one (like $take_order$). A trace then describes a sequence of events that happened for a special *case*, e.g. for a customer or a general object.

Definition 2 (Ordering relations [10]). *Let L be an event log over a set of activities T. For any $a, b \in T$ we introduce the following binary causal relations on T:*

- *$a >_L b$ if and only if a is directly followed by b somewhere in a trace σ in L, i.e. if a trace $\sigma = \langle t_1, \dots, t_n \rangle \in L$ and a number $i \in \{1, \dots, n-1\}$ exist, with $t_i = a$ and $t_{i+1} = b$*
- *$a \to_L b$ if and only if $a >_L b$ and $b \not>_L a$*
- *$a \leftarrow_L b$ if and only if $a \not>_L b$ and $b >_L a$*
- *$a \#_L b$ if and only if $a \not>_L b$ and $b \not>_L a$*
- *$a \|_L b$ if and only if $a >_L b$ and $b >_L a$.*

From these ordering relations $>_L$ will be of great interest for us, since we want to investigate the neighborhood of two events and decide whether two events directly following each other happen often (i.e. main behavior) or rarely (i.e. infrequent behavior). To do so, we need to know how often two events follow each other, which we can store in the *Correlation Matrix*:

Definition 3 (Correlation matrix). *Let L be an event log over T with $Start, End \notin T$. Further, take $T_S := T \cup \{Start\}$, $T_E := T \cup \{End\}$ and $\lambda := \langle\rangle$, the empty trace. We denote the number of times $a \in T_S$ is directly followed by $b \in T_E$ in all traces contained in L by $|(a, b)|_{>_L}$. The* correlation matrix *for an event log L is defined as the square matrix $C^L : T_S \times T_E \to \mathbb{N}_0$ with*

$$C^L_{a,b} = \begin{cases} |(a,b)|_{>_L} & \text{if } a, b \in T \\ \sum_{\sigma \in L \setminus \{\lambda\}, first(\sigma) = b} L(\sigma) & \text{if } a = \text{Start}, b \in T \\ \sum_{\sigma \in L \setminus \{\lambda\}, last(\sigma) = a} L(\sigma) & \text{if } a \in T, b = \text{End} \\ L(\lambda) & \text{if } a = \text{Start}, b = \text{End}. \end{cases}$$

Recall that L is a multiset and hence, for any $\sigma \in L$, $L(\sigma)$ denotes the number of times the trace σ is contained in L.

Apart from the Correlation Matrix, we can store the information related to which events are directly followed by one another in the *Directly Follows Graph*, which is a simple directed graph containing all events as vertices. Two vertices u and v are connected by an edge (u, v) if somewhere in the event log u is directly followed by v. The weight on the edge shows how often this happens in the event log.

Definition 4 (Directly Follows Graph). *Let L be an event log over a set of activities T. The* Directly Follows Graph *(DFG) for L is a weighted directed graph $G = (V, E, w)$ with*

- $V := T \cup \{Start, End\}$,
- $E := >_L \cup \{(Start, t) \mid \exists \sigma \in L : t = first(\sigma)\}$
  $\cup \{(t, End) \mid \exists \sigma \in L : t = last(\sigma)\}$ *and*
- $w : E \to \mathbb{N}_0$, $w((a, b)) := C^L_{a,b}$.

An example for a Directly Follows Graph will be given in Figure 1 in Section 4, where we first need the DFG. A useful property of the DFG is that it can easily be translated to a language-equivalent Petri-net. Leemans et al. [5] showed that if the DFG is *sound*, the respective language-equivalent Petri-net is also sound.

Definition 5 (Soundness of the Directly Follows Graph [5]). *Let $G = (V, E, w)$ be a DFG for an event log L. G is* sound *if every node $v \in V$ is on a path from $Start$ to End, i.e.*

$$\begin{aligned}\forall v \in V : \exists u_1, \ldots, u_n : & \; u_1 = Start \wedge \\ & \; u_n = End \wedge \\ & \; \exists j \in \{1, \ldots n\} : u_j = v \wedge \\ & \; \forall i \in \{1, \ldots, n-1\} : (u_i, u_{i+1}) \in E\end{aligned}$$

As a running example for this paper we take a set of activities $T = \{a, b, c, d\}$ and an artificial event log L which is defined as

$$L = [\langle a, b, c, b\rangle^{100}, \langle a, c, b\rangle^{50}, \langle d, b, d\rangle^{100}, \langle b, e\rangle^{1000}, \langle d, e\rangle^{1000}, \langle f, g, f, g, f, g\rangle^{100}].$$

To construct the Correlation Matrix, we simply count how often an event $e_1 \in T$ is followed by an event e_2 in the event log and add this quantity to the Correlation Matrix in row e_1 and column e_2.

	End	a	b	c	d	e	f	g
$Start$	0	150	1000	0	1100	0	100	0
a	0	0	100	50	0	0	0	0
b	150	0	0	100	100	1000	0	0
c	0	0	150	0	0	0	0	0
d	100	0	100	0	0	1000	0	0
e	2000	0	0	0	0	0	0	0
f	0	0	0	0	0	0	0	300
g	100	0	0	0	0	0	200	0

The artificial event $Start$ (End) is used to record how often an event $e \in T$ is the first (last) event in a trace of L. The Correlation Matrix is very handy to calculate values needed for the execution of hypothesis tests, which are described in the next section.

3. Hypothesis Tests

Rather recently, Petrak et al. [7] showed in their article that hypothesis tests can be used to determine whether the direct neighborhood of two events that was observed in the event log should be classified as infrequent behavior, i.e. *noise*, or not. We shortly revisit this idea, simplify it slightly, and give some simple examples that show how this method differs from simply considering the direct neighborhood that is the least frequent in the event log as noise.

Generally speaking, hypothesis tests can be used to check whether a certain hypothesis is valid with a "sufficiently high" probability. Such a hypothesis makes a statement about a certain property of the objects in a population. As a simple example, take the set of all researchers in the field of process mining as the population and the statement *"At least 80% of researchers in the field of process mining like process discovery"*. Since this is a statement that can't be proved or falsified without asking every single researcher in the population (which would be a rather costly operation) one could easily doubt the correctness of this statement. So next to the already stated *Null-Hypothesis* H_0, we formulate an *Alternative Hypothesis* H_1 stating the contrary: *"Less than 80% of researchers in the field of process mining like process discovery"*. To check which of the two hypotheses is correct, we take a small sample of the population and check how many researchers of this population like process discovery. We then perform a left-sided hypothesis test to check whether this sample implies that the Null-Hypothesis H_0 or the Alternative Hypothesis H_1 is true.

In general, let $p \in [0, 1]$ be an unknown probability (in our example the actual percentile of researchers who love process discovery) and $p_0 \in [0, 1]$ a parameter given by the user (in our example $p_0 = 0.8$). Using data from a sample of the population, a left-sided hypothesis test can be used to decide whether $p \in [0, p_0[$ or $p \in [p_0, 1]$ is true, i.e. whether the *Null Hypothesis* H_0: $p \geq p_0$ or the *Alternative Hypothesis* H_1: $p < p_0$ is true. Since a hypothesis test comes to this decision based on a sample of the full data, there is a certain probability that the wrong decision will be made. The probability that this happens can be estimated, making it possible to formulate rather reliable statements about which of the two hypotheses holds.

The probability that we chose H_1 when in reality H_0 is true is called the α-error; the probability that we chose H_0 when in reality H_1 is true, is called β-error. In the context of this paper, we are especially interested in the α-error, which can be bounded by a given α when using hypothesis tests. We then want to find a value k such that for a random variable X (which can be understood as the number of elements in a random sample that fulfill the property defined in H_0) $P(X \leq k \mid H_0 \text{ is true}) \leq \alpha$ is true. The values of k for which this inequality holds can be determined by using the density-function of the given probability distribution. However, since this computation is quite time-consuming, we use an approximation for k if said probability distribution is a binomial distribution and the standard-deviation $\sigma_n := \sqrt{n \cdot p_0 \cdot (1 - p_0)}$ satisfies $\sigma_n > 3$. In this case, we approximate k by

$$k = \lceil np_0 - \sigma_n \cdot u_{1-\alpha} \rceil, \tag{1}$$

where $u_{1-\alpha}$ is the $(1 - \alpha)$-quantile of the standard normal distribution. A more formal description of hypothesis tests can be found in [11].

To detect noise in an event log, Petrak et al. [7] understand the neighborhood relation between two events as a binomial distribution and use hypothesis tests to check whether the sighting of two events $e_1, e_2 \in T$ directly following each other is main behavior (H_0 is true) or infrequent behavior (H_1 is true). To achieve this, they define the population as the pairs $(e, e') \in T$ that can follow each other in the process and where $e = e_1$ or $e' = e_2$ and view the event log as a sample where some of these direct neighborhoods were randomly drawn. Hence, the population is defined as:

$$P_{(e_1,e_2)} := \{(x, y) \in T_S \times T_E \mid (x = e_1 \vee y = e_2) \wedge |(x, y)|_{>L} > 0\},$$

which defines the sample size n as the number of events that can follow e_1 or can be followed by e_2 in the process:

$$n := |P_{(e_1,e_2)}| = \sum_{e \in T_E} |(e_1, e)|_{>L} + \sum_{e \in T_S} |(e, e_2)|_{>L} - |(e_1, e_2)|_{>L}.$$

With the user-given constants $1 - p_0$ and α they then execute a right-sided hypothesis test, which means, for a pair of events (e_1, e_2), they estimate k by $\lceil np_0 - \sigma_n \cdot u_{1-\alpha} \rceil$ and decide for H_0 when $n - |(e_1, e_2)|_{>L} < k$. Since this is equivalent to estimate k by the formula in Equation 1 and deciding for H_0 when $|(e_1, e_2)|_{>L} > k$, we can instead perform a left-sided hypothesis test with the probability p_0 as described at the start of this section.

With this idea we can calculate the critical value k for which $P(X \leq k \mid H_0 \text{ is true}) \leq \alpha$ holds and check afterwards whether $|(e_1, e_2)|_{>L} > k$. If this is true, we accept the Null Hypothesis H_0 and consider e_2 directly following e_1 as main behavior. Otherwise, we reject the Null Hypothesis, which means we accept the Alternative Hypothesis H_1 and consider e_2 directly following e_1 as infrequent behavior. However, if $\sigma_n \leq 3$, we need to calculate the critical value by evaluating the binomial distribution and finding a k for which the integral of the density function from 0 to k is less than $1 - \alpha$. For a more detailed description of this idea, see [7] and [11]. Using these ideas leads to the procedure shown in algorithm 1.

For our running example, we set $p_0 = 0.05$ (so we consider a pair of direct neighbors as infrequent if this behavior affects less than 5% of the population) and $\alpha = 0.05$ (so we accept that the probability for classifying main behavior as infrequent is at most 5%). We show the decision for whether a pair of direct neighbors is main or infrequent behavior on the following two examples:

Check $(a, c) \in >_L$:

	End	*a*	*b*	*c*	*d*	*e*	*f*	*g*
Start	0	150	1000	0	1100	0	100	0
a	0	0	100	50	0	0	0	0
b	150	0	0	100	100	1000	0	0
c	0	0	150	0	0	0	0	0
d	100	0	100	0	0	1000	0	0
e	2000	0	0	0	0	0	0	0
f	0	0	0	0	0	0	0	300
g	100	0	0	0	0	0	200	0

$n = 250$

$\sigma_n \approx 3.4 > 3$

$\Rightarrow k = 7$

$|(a, c)|_{>L} = 50 > 7 = k$

$\Rightarrow (a, c) \in >_L$ is main behavior.

Algorithm 1 Detecting main and infrequent neighbors

Input: $p_0 \in [0,1]$, $\alpha \in [0,1]$, $L : T^* \to \mathbb{N}_0$
Output: $main, infreq \subseteq (T \cup \{Start, End\})^2$

```
main ← ∅
infreq ← ∅
for (e_1, e_2) ∈ T ∪ {Start, End} do
    n ← Σ_{e∈T_E} |(e_1, e)|_{>L} + Σ_{e∈T_S} |(e, e_2)|_{>L} − |(e_1, e_2)|_{>L}
    σ_n ← √(n · p_0 · (1 − p_0))
    if σ_n > 3 then
        k ← ⌈n p_0 − σ_n · u_{1−α}⌉
    else
        sum = 0
        while sum < 1 − α do
            sum ← sum + (n choose k) · p_0^k · (1 − p_0)^{n−k}
            k ← k + 1
        end while
    end if
    if |(e_1, e_2)|_{>L} > k then                    ▷ k > 0, so |(e_1, e_2)|_{>L} > 0.
        main ← main ∪ {(e_1, e_2)}
    else
        infreq ← infreq ∪ {(e_1, e_2)}
    end if
end for
return main, infreq
```

It is notable that in this example the direct neighborhood of the events a and c has the lowest frequency, but the result of the hypothesis test implies that a being directly followed by c is main behavior. This is due to the fact that a is only followed by b or c and c is only preceded by a or b. The number of sightings where b precedes c or b follows after a is rather small, so that the neighborhood between a and c can be considered as main behavior.

Check $(b, d) \in >_L$:

	End	a	b	c	d	e	f	g
$Start$	0	150	1000	0	1100	0	100	0
a	0	0	100	50	0	0	0	0
b	150	0	0	100	100	1000	0	0
c	0	0	150	0	0	0	0	0
d	100	0	100	0	0	1000	0	0
e	2000	0	0	0	0	0	0	0
f	0	0	0	0	0	0	0	300
g	100	0	0	0	0	0	200	0

$n = 2450$
$\sigma_n \approx 10.7 > 3$
$\Rightarrow k = 105$
$|(b,d)|_{>_L} = 100 \leq 105 = k$

$\Rightarrow (b, d) \in >_L$ is infrequent behavior.

Executing the hypothesis test for every pair of events leads to the result that the pairs (b, d), (b, End), (d, b), (d, End) and $(Start, f)$, of direct neighbors are infrequent. Petrak et al. propose to simply delete these direct neighborhood relations from the footprint and use the footprint that was constructed in this way to mine a process model.

4. Using the results to construct a DFG

We use the approach presented by Petrak et al., but construct a Directly Follows Graph (see Definition 4) instead of a footprint. Gaining a Petri-net from a DFG is rather simple, and we can easily check the soundness of a DFG by performing two Depth First Searches, which gives us the possibility to easily construct a DFG that is sound.

By applying the hypothesis tests to the event log with Algorithm 1 we immediately get a set $M \subseteq (T \cup \{Start, End\})^2$ of direct neighbors that are part of the main behavior of the event log and a set $I \subseteq (T \cup \{Start, End\})^2$ of direct neighbors that are infrequent behavior. Obviously, no direct neighborhood can be both main and infrequent behavior, so $M \cap I = \emptyset$. Let $G = (V, E)$ be the DFG for our event log. Since we only add behavior to M or I if we have seen the direct neighborhood at least once in the event log, $M \subseteq E$ and $I \subseteq E$ hold. On the other hand, due to the definition of E, there is no pair of events in $M \cup I$ that is not present in E, since E contains every pair of direct neighbors that was observed in the event log. Therefore, we can conclude that $M \cup I = E$. The DFG for our running example is shown in Figure 1, the black edges are those of the set M and the highlighted edges are those of the set I.

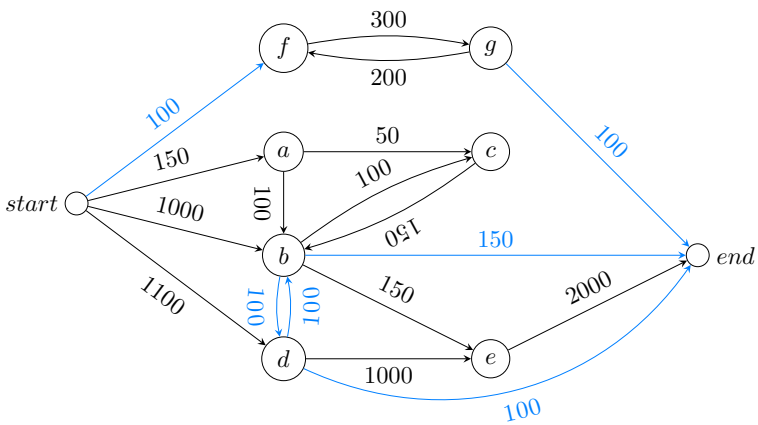

Figure 1: The DFG for the event log of our running example L. Blue edges were classified as infrequent behavior, black edges were classified as main behavior by the hypothesis tests.

When deleting edges from the DFG, we need to be careful, since the deletion of the whole set I may lead to a result that is not sound according to Definition 5. This is also the case for the DFG in Figure 1, since without all the highlighted edges, the events f and g do not lie on a path

from $Start$ to End. A process model constructed directly from such a DFG is not sound, so the goal is to delete as many infrequent edges as possible without sacrificing the soundness of the DFG. Deleting every highlighted edge except the edges $(Start, f)$ and (g, End) leads to the result shown in Figure 2.

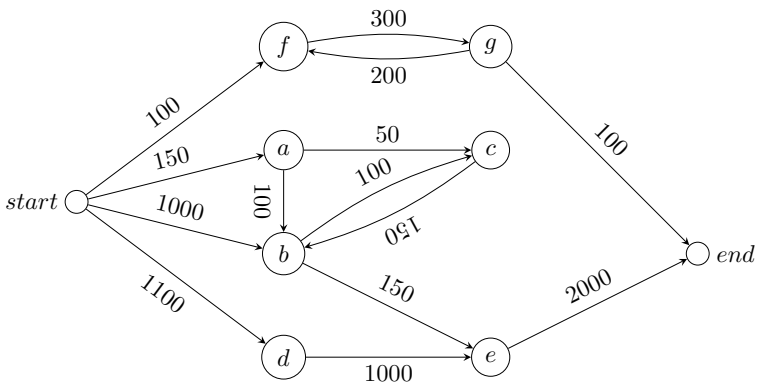

Figure 2: The modified DFG for our running example log L, where infrequent edges that were not necessary for the soundness were deleted.

In general, it is not always clear which set $J \subseteq I$ should be deleted from the DFG, since the deletion of an edge e could lead to a situation where other edges are needed for soundness, whereas said edges would not be needed if e wasn't deleted. Finding the "best" subset of I to delete from the DFG is a difficult problem, since it is not clear when one subset is better than another. If we define a function $f : \mathcal{P}(I) \to \mathbb{N}_0$ which maps every subset of I to a numerical value, where a high value indicates that the subset is a better candidate for deletion than another candidate with a lower value, we have an optimization problem at hand: Maximize $f(J)$ where $G' = (V, E \setminus J)$ is a sound DFG and $J \subseteq I$. As a default for f we propose $f(J) := |J|$, so that edge sets that contain more infrequent edges are preferred for deletion. A naive implementation that solves this problem can be found in Algorithm 2.

Algorithm 2 Deleting infrequent edges from the DFG

Input: $G = (V, E)$, $M, I \subseteq E$ with $M \cup I = E$ and $M \cap I = \emptyset$, $f : \mathcal{P}(I) \to \mathbb{N}_0$
Output: $G' = (V, E')$

```
J_opt ← ∅
for each J ⊆ I do
    G'' ← (V, E \ J)
    if f(J) > f(J_opt) and G'' is sound then
        J_opt ← J
    end if
end for
return (V, E \ J_opt)
```

This algorithm needs $O(2^{|I|} \cdot (|V| + |E|))$ time and is therefore exponential in time, but since

the number of infrequent edges, and therefore the cardinality $|I|$ of I, is rather small in the most cases, we accept this naive implementation, since it allows for an easy exchange of the user-given function f that should be optimized, and is easy to read. In our running example, this algorithm leads to the sound DFG shown in Figure 2.

5. Handling of Loops

As can be seen in the example of Figure 1, loops in the DFG can have a high impact on the population for our hypothesis tests. In this example, executing the loop between f and g some times artificially increases the number of times when f was preceded by another event than $Start$ and the number of times when after g followed another event than End. Even though the edges $(Start, f)$ and (g, End) are only encountered in the same trace as the edges (f, g) and (g, f), the repetition of this loop leads to the effect that $(Start, f)$ and (g, End) are classified as infrequent behavior. In the example of Figure 1 this is not a problem, since these two infrequent edges are not deleted for the sake of soundness. However, the following simple event log shows that we are not always in such a lucky situation:

$$L_{\circlearrowleft} = [\langle a, \underbrace{b, \ldots, b}_{51 \text{ times}}, d\rangle^{10}, \langle a, b, c, d\rangle^{40}, \langle a, c, d\rangle^{100}]$$

Executing hypothesis tests on this log leads to the classification of the direct neighborhood between b and d as infrequent.

Check $(b, d) \in >_{L_{\circlearrowleft}}$:

	End	a	b	c	d
$Start$	0	150	0	0	0
a	0	0	50	100	0
b	0	0	500	40	10
c	0	0	0	0	140
d	150	0	0	0	0

$n = 690$

$\sigma_n \approx 5.7 > 3$

$\Rightarrow k = 26$

$|(b, d)|_{>_L} = 10 \leq 26 = k$

$\Rightarrow (b, d) \in >_{L_{\circlearrowleft}}$ is infrequent behavior.

Every other direct neighborhood is classified as main behavior by the hypothesis tests, so we get the DFG for $\mathcal{L}$ that is shown in Figure 3.

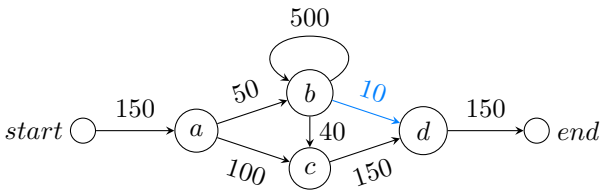

Figure 3: The DFG for the event log $L_{\circlearrowleft}$ with the only infrequent edge highlighted in blue.

Deletion of the highlighted edge would not violate the soundness of the DFG, but doing so would remove the entire behavior of a being followed by one or more bs, which is followed by a

single d. This is a problem, since the neighborhood (b, d) would not be classified as infrequent behavior if the event log would feature fewer iterations of the b-loop:

Check $(b, d) \in >_{L_{\circlearrowleft}}$:

	End	a	b	c	d
$Start$	0	150	0	0	0
a	0	0	50	100	0
b	0	0	10	40	10
c	0	0	0	0	140
d	150	0	0	0	0

$n = 200$

$\sigma_n \approx 3.08 > 3$

$\Rightarrow k = 5$

$|(b, d)|_{>_L} = 10 > 5 = k$

$\Rightarrow (b, d) \in >_{L_{\circlearrowleft}}$ is main behavior.

To eliminate this problem, we propose a preprocessing step for the hypothesis tests to reduce the number of loop iterations if there is a loop present in the process. To do so, we alter the traces of our event log that contain such a loop for the hypothesis tests. The idea is to construct a DFG for this single trace and find a path from $Start$ to End that cycles less frequently through the loop in the trace, if this is possible. To do so, we firstly introduce the notion of a *Trace DFG*:

Definition 6 (Trace DFG). *Let $\sigma = \langle t_1, \ldots, t_n \rangle$ be a trace. The* Trace DFG *for σ is a directed multigraph $G = (V, E)$ with*

- $V := \{Start, End\} \cup \{t_1, \ldots, t_n\}$ *and*
- $E : (T \cup \{Start, End\})^2 \to \mathbb{N}_0$,

$$E((t_1, t_2)) = \begin{cases} 1 & \text{if } t_1 = Start \text{ or } t_2 = End \\ |\{i \mid t_i = t_1 \wedge t_{i+1} = t_2\}| & \text{otherwise} \end{cases}$$

For a trace σ, the Trace DFG is therefore the DFG of the log $[\sigma]$, where the edge weights of said DFG are interpreted as multiple edges. The Trace DFG for the trace $\langle a, b, c, d \rangle$ for example is a simple path, the Trace DFG for the trace $\langle a, \underbrace{b, \ldots, b}_{51 \text{ times}}, d \rangle$ is shown in Figure 4.

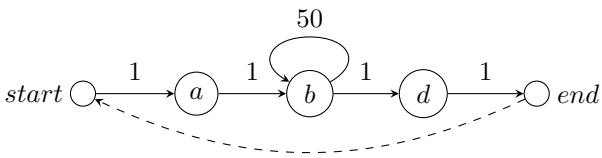

Figure 4: The Trace DFG for $\langle a, b, \ldots, b, d \rangle \in \mathcal{L}$. Edge weights denote the multiplicity of edges, the artificial dashed edge (which is not part of the Trace DFG) makes the DFG eulerian.

To reduce the number of loop iterations in the Trace DFG we first introduce a new, artificial edge from End to $Start$, as shown by the dashed dart in Figure 4. The resulting multigraph G is eulerian, which means there is a eulerian cycle in G.

Theorem 1. *Let $\sigma \in T^*$ be a trace and $G = (V, E)$ the Trace DFG of σ. Then, the graph $G' = (V, E \cup \{(End, Start)\}$ is eulerian, i.e. contains a eulerian cycle.*

Proof. By construction of the Trace DFG, σ is a path from $Start$ to End that uses every edge in G exactly once. With the artificial edge $(End, Start)$ in G' this path becomes a cycle that uses every edge in G' exactly once, i.e. a eulerian cycle.

Since this extended Trace DFG G' is eulerian, we can find the shortest cycle through G' that uses every type of edge in G' at least once, but no more than the multiplicity of the edges allow. The problem of finding this shortest cycle is known as the Chinese Postman Problem, which was introduced by Edmonds et al. [9] and can be solved in $O(|V|^3)$ time. In the example for $L_{\circlearrowright}$, the shortest cycle is $(start, a, b, b, d, end, start)$, which cycles only once through the loop at the vertex b. From this cycle we can easily construct the shortened trace $\langle a, b, b, d\rangle$ that features less loop iterations than the original trace and hence does not disrupt the hypothesis tests. We therefore use this trace instead of the original trace to execute the hypothesis tests and find that the neighborhood (b, d) is main behavior. However, when constructing the DFG for $L_{\circlearrowright}$, we calculate the multiplicities based on the original (i.e. not shortened) traces.

6. Evaluation

We implemented the algorithm described in this paper in `python`, version `3.8`, using the process mining library `pm4py` [12]. The latter provides functions to calculate the fitness, precision, generalization and simplicity for a given Petri net. To calculate these quality-measures, we used the following techniques provided by `pm4py`:

- **Fitness:** The alignment-based fitness is computed and the percentage of completely fit traces is returned, as well as the average fitness value of the single traces,
- **Precision:** The Align-ETConformance is computed and returned, see [13],
- **Generalization:** Computed as described in [14],
- **Simplicity:** Computed as described in [15].

To check the Petri net for soundness, the external tool WOFLAN [16] is used. Aside from `pm4py` there are other tools, like Entropia [17], that are also interesting for efficient and robust conformance checking. We plan to investigate these tools in the future and use them to further evaluate our results.

We chose some real-life event logs that are often used to evaluate new process discovery techniques, to test and compare our implementation to other mining-techniques. We used

- `BPI_Challenge_2012.xes` (**BPI12**), which can be found at [18],
- `DomesticDeclarations.xes` (**DD**),
- `InternationalDeclarations.xes` (**ID**),
- `PermitLog.xes` (**PL**),
- `PrepaidTravelCost.xes` (**PTC**),
- `RequestForPayment.xes` (**RFP**), all of which can be found at [19].

Table 1
Characteristics of the event logs that were chosen for the evaluation.

	BPI12	DD	ID	PL	PTC	RFP
Number of unique traces	4 366	99	753	1 478	202	89
Total number of traces	13 087	10 500	6 449	7 065	2 099	6 886
Number of cycles	43 566	121	585	3 842	113	80
Longest Trace Length	175	24	27	90	21	20
Shortest Trace Length	3	1	3	3	1	1
Average Trace Length	20.04	5.37	11.19	12.25	8.69	5.34

Table 1 gives an overview over some important characteristics of these event logs.

First, we compared our approach with the fixed parameters $p_0 = 0.05$ and $\alpha = 0.05$ with the following well-known mining approaches that consider outliers in the event log:

- The *Inductive Miner infrequent* (IMi) [6] creates a process tree out of the DFG for an event log and creates a Petri net from that process tree. In a preprocessing step, all edges that are "too infrequent" are removed, by going through all events e and deleting all outgoing edges of e with a frequency less than k times the frequency of the strongest outgoing edge of e, where k is a user-chosen parameter between 0 and 1. We chose 0.2 as the parameter for IMi.
- The *Directly Follows Miner* (DFM) [5] creates a DFG and scans it for infrequent behavior by determining the edges with the least frequency. Then all traces that feature the direct neighborhood modeled with an infrequent edge are removed from a copy of the event log. This step is repeated until at most a fraction of t traces were removed, where t is a user-chosen threshold between 0 and 1. Then, the DFG for the new event log is computed (or the old DFG is adjusted accordingly) and a Petri net is constructed from the DFG. We chose 0.1 as the parameter for DFM.

We also compared our approach to the technique presented in [7], where hypothesis tests are executed, and all infrequent edges are deleted from the DFG without the goal to maintain soundness. For this approach, we chose $p_0 = 0.95$ and $\alpha = 0.05$, so the hypothesis tests are performed exactly like with the new technique.

Since the DFM and our approach result in a DFG instead of a Petri net, we also need to convert the DFG into a Petri net to compute the quality measures as implemented in `pm4py`. To do so we constructed a labeled Petri net $N = (P, T, F, l)$ from a DFG $G = (V, E, w)$ as follows:

- For each node $v \in V$ a place $v \in P$ is added.
- For each edge $(u, v) \in V \times (V \setminus \{End\})$, a transition $t_{u,v} \in T$ with $l(t_{u,v}) = v$ is added.
- For each edge $(u, End) \in V \times \{End\}$, we also add an edge $t_{u,End} \in T$. However, this is a silent transition, i.e. $l(t_{u,End}) = \tau$.
- Finally, for each transition $t_{u,v} \in T$ two arcs are added: $(u, t_{u,v}) \in F$ and $(t_{u,v}, v) \in F$.

The results of executing the algorithms and the conformance checking methods on the resulting Petri nets can be found in Figure 5. For all evaluated methods and logs, Fitness and Generalization were similarly good. Simplicity also showed consistently good values, except for

a few very good results from the DFM. The percentage of fitting traces was consistently very good for DFM, while the other techniques achieved variously good values for different logs. The largest differences between the methods were observed for the Precision and the F-Score. Here, the results for different logs vary greatly for the DFM, while IMi achieves rather low results. Both of our new approaches achieve very good values most of the time in these two categories.

It is notable that the approach of [7] (which we call HT) always leads to a better precision than our new approach, which is not surprising considering that we leave some infrequent edges in the graph to maintain soundness. For example, if there is a single trace featuring much infrequent behavior, from which almost everything is deleted from the DFG except for an edge e, it is possible to use this edge e in the resulting Petri Net, even though it does not fit to any behavior seen in the event log. Due to the higher precision, the F-Score is also better than in our approach. Further, the simplicity of the net mined by HT is often much higher than in the new approach, simply because more edges are deleted. However, contrary to our new approach, HT never led to a sound net for any of the tested event logs.

IMi scores similary to our approach, except for precision and therefore F-Score. This is due to the fact that IMi classifies edges as infrequent with a rather *local* argument – the frequency of an edge is compared to the frequencies of all other outgoing edges, but not to the frequencies in the full graph. Hence, IMi may often delete single edges from many traces. For example, for the very simple event log $[\langle a, b, c\rangle^l]$, it can easily be forced to delete the edge (b, c) by introducing a trace $\langle x, b, y\rangle^{\frac{l}{k}+1}$ to the event log. Then, the edge (b, c) in the DFG becomes infrequent compared to the other outgoing edge (b, y) and is therefore deleted. Then, the trace $\langle a, b, c\rangle$ can not be replayed, which leads to a worse fitness, but the trace $\langle a, b, y\rangle$ can be replayed, even though it is not present in the event log.

With the user-chosen threshold that affects the behavior of the DFM, we have direct control over the fitness of the mined model, so this metric features always very good values. However, since DFM deletes only traces and not single edges, it seems odd that the precision of DFM is as low as it is for some event logs. This may be associated with the fact that in the DFG there is exactly one vertex for every event present in the event log, and therefore some traces that share an event name but have nothing else in common lead to a Petri net where traces can be replayed that are not part of the event log. However, our approach shares the same problem concerning the DFG, which makes it unclear why the precision of our approaches have a much higher precision most of the time. To investigate this behavior further could be an interesting task for future work.

When comparing the simpler version of our method to the one with loop reduction, it can be seen that both versions achieve similar results, which can be explained by the structure of the chosen event logs. It may be interesting to investigate further how the two variants differ on event logs with many loops.

During the execution of the different algorithms, the times required to construct a model in each case were measured. These do not include the calculation of the metrics that were afterwards calculated on it. In Table 2, all execution times are given in seconds. Hence, for all event logs and algorithms evaluated, a result could be calculated within a few seconds.

Table 2
The execution times in seconds for each investigated approach and each chosen event log.

	BPI12	DD	ID	PL	PTC	RFP
HTs	00.87397 s	00.09229 s	00.20281 s	02.73899 s	00.07453 s	00.06550 s
HTl	02.69569 s	00.10818 s	00.36392 s	01.00158 s	00.10343 s	00.07757 s
HT	05.493242 s	00.09222 s	00.18860 s	00.34547 s	00.05466 s	00.06677 s
IMi	12.15239 s	01.67628 s	03.43188 s	07.16169 s	01.54955 s	01.86956 s
DFM	01.99909 s	00.14701 s	00.31106 s	00.68258 s	00.05291 s	00.09879 s

7. Conclusion

In this paper, we revisited the use of hypothesis tests of Petrak et al. [7] to detect infrequent neighborhood relations in a given event log. We proposed a simpler variant for these tests in the form of a left-sided hypothesis test and changed the output to a Directly Follows Graph instead of a footprint, in order to be able to selectively remove infrequent behavior without destroying the soundness of the resulting DFG. Furthermore, we showed theoretically that high loop iterations in an event log can lead to undesired results of the hypothesis tests and solved this problem by implementing a preprocessing step that reduces the number of loop iterations to an amount small enough that the sample size used for the hypothesis tests is not bloated by the loop. The proposed techniques are very easy to understand and implement. We compared both the hypothesis tests without loop-shortening and the hypothesis tests with loop-shortening with the results of the old technique using hypothesis tests [7], the Inductive Miner infrequent [6] and the Directly Follows Miner [5] and found that our results for fitness, precision, generalization, simplicity and F-score are rather consistent and high – also compared to these techniques.

Further research should include the evaluation of logs with high loop iterations to verify that the shortening of loops is not only reasonable in theory, but also in practice. It is interesting how often the loop-shortening is relevant in real-life logs and under what circumstances (concerning the underlying process) the necessity of loop-shortening is probable. Also, there are some more techniques that outliers before mining a process model [20, 21, 22], some of which also use statistical ideas. It would be very interesting to see how our approach competes to these techniques. Since the DFG loses information on concurrency, a preprocessing step to detect concurrency may be a topic of further research – as well as the investigation on the existence of event logs that lead to bad results with our approach due to concurrency.

References

[1] S. J. J. Leemans, D. Fahland, W. M. P. van der Aalst, Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour, in: Business Process Management Workshops - BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers, volume 171 of *Lecture Notes in Business Information Processing*, 2013, pp. 66–78.

[2] C. W. Günther, W. M. P. van der Aalst, Fuzzy Mining – Adaptive Process Simplification Based on Multi-perspective Metrics, in: Business Process Management, 2007, pp. 328–343.

[3] A. Weijters, W. Aalst, van der, A. Alves De Medeiros, Process mining with the HeuristicsMiner algorithm, BETA publicatie : working papers, Technische Universiteit Eindhoven, 2006.

[4] A. J. M. M. Weijters, J. T. S. Ribeiro, Flexible Heuristics Miner (FHM), in: 2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM), 2011, pp. 310–317.

[5] S. J. J. Leemans, E. Poppe, M. T. Wynn, Directly Follows-Based Process Mining: Exploration & a Case Study, in: International Conference on Process Mining, ICPM 2019, Aachen, Germany, June 24-26, 2019, IEEE, 2019, pp. 25–32.

[6] S. J. J. Leemans, D. Fahland, W. M. P. van der Aalst, Discovering Block-Structured Process Models from Incomplete Event Logs, in: G. Ciardo, E. Kindler (Eds.), Application and Theory of Petri Nets and Concurrency - 35th International Conference, PETRI NETS 2014, Tunis, Tunisia, June 23-27, 2014. Proceedings, volume 8489 of *Lecture Notes in Computer Science*, Springer, 2014, pp. 91–110.

[7] L. Petrak, R. Lorenz, Detecting Infrequent Behavior in Event Logs using Statistical Inference, in: Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data 2020, June 24, 2020, volume 2625 of *CEUR Workshop Proceedings*, 2020, pp. 33–48.

[8] J. Carmona, B. F. van Dongen, A. Solti, M. Weidlich, Conformance Checking - Relating Processes and Models, Springer, 2018. URL: https://doi.org/10.1007/978-3-319-99414-7. doi:10.1007/978-3-319-99414-7.

[9] J. R. Edmonds, E. L. Johnson, Matching, Euler tours and the Chinese postman, Math. Program. 5 (1973) 88–124.

[10] W. M. van der Aalst, B. F. van Dongen, Discovering Petri Nets from Event Logs, in: Transactions on Petri Nets and Other Models of Concurrency VII, Springer, 2013, pp. 372–422.

[11] G. Hornsteiner, Daten und Statistik, Eine praktische Einführung für den Bachelor in Psychologie und Sozialwissenschaften. Berlin/Heidelberg (2012).

[12] A. Berti, S. J. Van Zelst, W. van der Aalst, Process Mining for Python (pm4py): bridging the gap between process-and data science, arXiv preprint arXiv:1905.06169 (2019).

[13] A. Adriansyah, J. Munoz-Gama, J. Carmona, B. Dongen, W. Aalst, Measuring precision of modeled behavior, Information Systems and e-Business Management 13 (2014). doi:10.1007/s10257-014-0234-7.

[14] J. Buijs, B. Dongen, W. Aalst, Quality Dimensions in Process Discovery: The Importance of Fitness, Precision, Generalization and Simplicity, International Journal of Cooperative Information Systems 23 (2014) 1440001. doi:10.1142/S0218843014400012.

[15] F. R. Blum, Metrics in process discovery, 2015.
[16] W. M. P. van der Aalst, Woflan: A Petri-net-based Workflow Analyzer, Systems analysis modelling simulation 35 (1999) 345–357. URL: https://publications.rwth-aachen.de/record/714622.
[17] A. Polyvyanyy, H. Alkhammash, C. D. Ciccio, L. García-Bañuelos, A. A. Kalenkova, S. J. J. Leemans, J. Mendling, A. Moffat, M. Weidlich, Entropia: A Family of Entropy-Based Conformance Checking Measures for Process Mining, CoRR abs/2008.09558 (2020). URL: https://arxiv.org/abs/2008.09558. arXiv:2008.09558.
[18] B. van Dongen, BPI Challenge 2012, 2012. URL: https://data.4tu.nl/articles/dataset/BPI_Challenge_2012/12689204/1. doi:10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f.
[19] B. van Dongen, BPI Challenge 2020, 2020. URL: https://data.4tu.nl/collections/BPI_Challenge_2020/5065541/1. doi:10.4121/uuid:52fb97d4-4588-43c9-9d04-3604d4613b51.
[20] R. Conforti, M. L. Rosa, A. H. t. Hofstede, Filtering Out Infrequent Behavior from Business Process Event Logs, IEEE Transactions on Knowledge and Data Engineering 29 (2017) 300–314. doi:10.1109/TKDE.2016.2614680.
[21] X. Lu, D. Fahland, W. M. van der Aalst, Interactively Exploring Logs and Mining Models with Clustering, Filtering, and Relabeling, in: BPM, 2016.
[22] D. Brons, R. Scheepens, D. Fahland, Striking a new Balance in Accuracy and Simplicity with the Probabilistic Inductive Miner, in: 2021 3rd International Conference on Process Mining (ICPM), 2021, pp. 32–39. doi:10.1109/ICPM53251.2021.9576864.

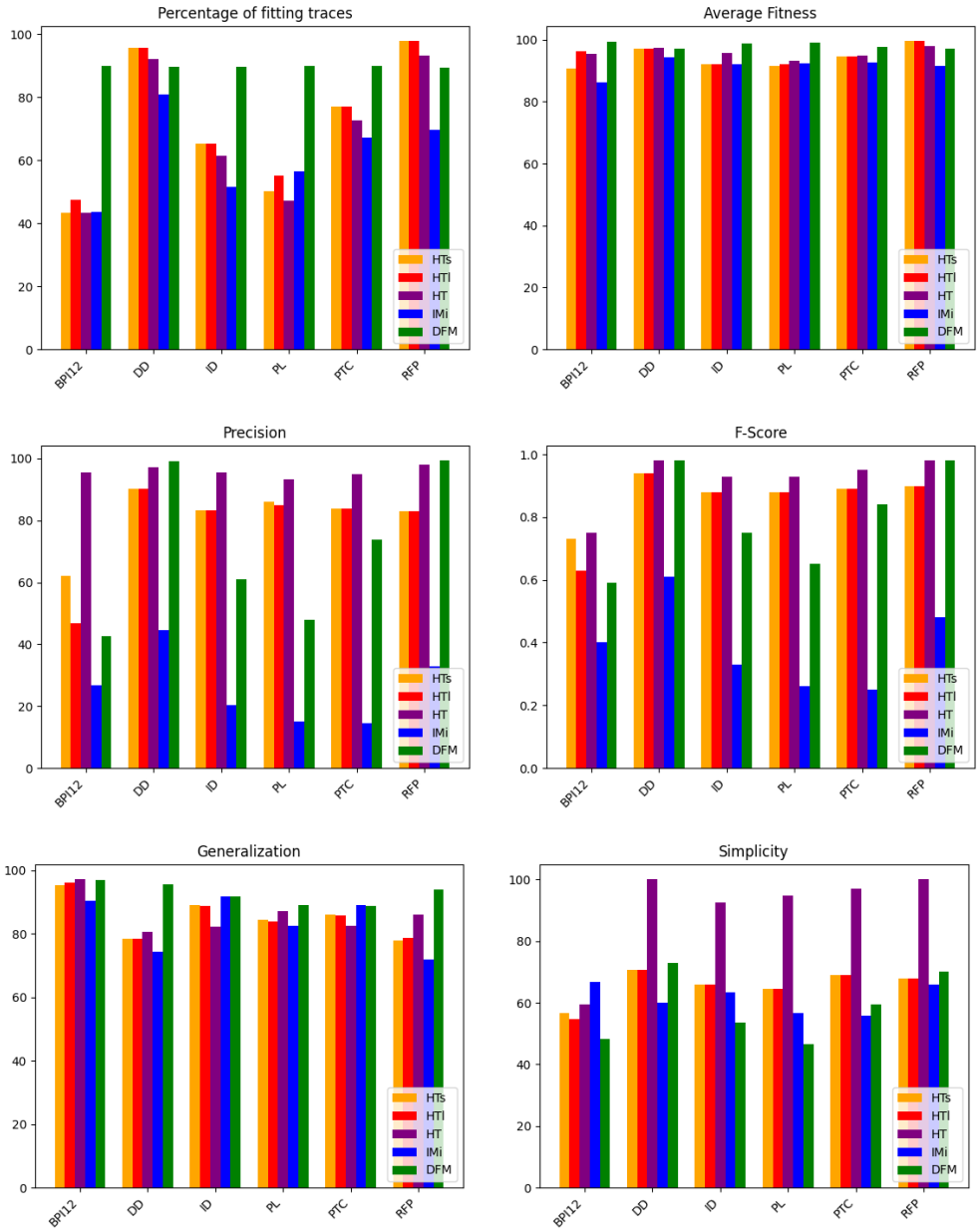

Figure 5: Comparison of the resulting quality-metrics for the approach described in this paper, using hypothesis tests and maintaining soundness (HTs), the same approach using loop-reduction (HTl) the old technique using hypothesis tests (HT), the Inductive Miner infrequent (IMi) and the Directly Follows Miner (DFM).