# 2019 International Conference on Process Mining (ICPM)

# ICPM 2019

## Table of Contents



## Data Pre-Processing



## Automated Process Discovery

## Process Mining Applications



## Conformance Checking



## Data Analysis and Monitoring

## Predictive Monitoring

# Message from the General Chair
## ICPM 2019

Welcome to the first International Conference on Process Mining (ICPM 2019). We are very proud to be able to host this historic event in the beautiful city of Aachen. As the "godfather of process mining" it is great to see how the discipline is growing and maturing. Process mining emerged two decades ago as the glue between process science (BPM, WFM, OR, concurrency theory, etc.) and data science. The evidence-based nature of process mining helps to connect business and IT people and is also a field of research where the collaboration between industry and academia is very direct. Process mining techniques discovered in academia end up in commercial tools within a few years. Practical applications of commercial tools result in new scientific challenges. Although process mining tools are already widely used and have proven to be of great value, we are still at the start of this exciting development. Many of the known problems in process discovery, conformance checking, predictive analytics, and automated process improvement are still not solved satisfactorily. The range of potential applications in our modern data-driven society is so broad that spectacular growth is inevitable.

The confrontation between modeled behavior and observed behavior generates novel scientific problems that can only be solved by experts that are able to link process science and data science. ICPM 2019 has a scientific track presenting the research results presented in these proceedings. The industry day organized by Marc Kerremans (Gartner) includes a panel and invited talks by Gia-Thi Nguyen (Siemens), Michael Wiese (Ernst & Young), Piergiorgio Grossi (Credem Banca), Marc Gittler and Patrick Greifzu (DHL), Carsten Schöne (Merck), David Whyte (Canadian Financial Services), Bart Prudon (Medtronic), and Julian Lebherz (Deloitte).

The conference takes place in the Tivoli football stadium close to the city center of Aachen. This beautiful venue will provide a unique atmosphere with great views and excellent conference facilities. In the same week, also the 40th International Conference on Application and Theory of Petri Nets and Concurrency (PN 2019), the 19th International Conference on Application of Concurrency to System Design (ACSD 2019), and the workshop Algorithms & Theories for the Analysis of Event Data (ATAED) take place at the same location.

I would like to thank the members of the Process and Data Science (PADS) group at RWTH Aachen University for helping to organize this event. Thanks to all authors, presenters, reviewers, program committee members, contest organizers, and participants we are able to create this wonderful milestone in process mining history. Special thanks go to the program chairs of ICPM 2019: Josep Carmona (Universitat Politècnica de Catalunya, Spain), Mieke Jans (Hasselt University, Belgium), and Marcello La Rosa (The University of Melbourne, Australia). ICPM 2019 is supported by Celonis, Deloitte, ProcessGold, SAP, myInvenio, Fluxicon, Everflow, Minit, PuzzleData, PAFnow, Software AG, StereoLOGIC, BrightCape, Logpickr, Mehrwerk, QPR, KPMG, LanaLabs, Wintec, RWTH, Fraunhofer FIT, Gartner, the Alexander von Humboldt Foundation, the Deutsche Forschungsgemeinschaft, IEEE, and many more organizations. This broad support for ICPM illustrates the growing interest in process mining as a tool-independent discipline.

ICPM 2019 is the first instance of a new conference series that emerged from the IEEE Task Force on Process Mining. I am looking forward to the further development of our community.

Wil van der Aalst
***General Chair ICPM 2019***

# Message from the Program Chairs
## ICPM 2019

Welcome to the 1st International Conference on Process Mining, ICPM 2019.

We are thrilled to start this long-awaited conference series on Process Mining. ICPM is meant to be a multi-disciplinary, heterogeneous forum, where all aspects of the process mining field are covered: from scientific research describing how fundamental challenges are overcome, through to the applied research which shows disruptive applications of this emerging field. This first edition of the conference has set the direction to follow for the years to come, in order to consolidate the relatively-young research discipline of process mining.

The conference received 50 full paper submissions. Each paper was reviewed by at least three Program Committee members, plus a meta-review performed by a Program Chair. At the end of the process we accepted 18 excellent papers (acceptance rate of 36%). We believe we have compiled a solid program, split into sessions corresponding to major area of research within process mining, such as automated process discovery, conformance checking, predictive monitoring, data-preprocessing and analysis, and applications.

The conference started with a keynote by Wil van der Aalst, the initiator of the field and its most active evangelist. On his keynote, Wil gave a historical positioning of the field, and his personal predictions for its future developments, both in terms of research and applications. Moreover, we had an invited tutorial on two complementary angles for temporal analysis of process execution data, for online monitoring and improvement of complex systems. Also, we held a plenary panel on emerging applications of process mining.

Other activities were carried out in parallel with the scientific program. First, on Sunday, we held the Doctoral Consortium. The Consortium offered a great opportunity to our new PhD students to seek early feedback on their process mining research. To witness the rapid uptake of process mining in practice, the Industry Day was held on Tuesday, blended between the two research days on Monday and Wednesday. This event provided a rich showcase of process mining applications, through a series of industry keynotes where practitioners could directly report on their own success stories and lessons learnt. We also announced the winners of the Process Discovery Contest, the BPI Challenge, the Conformance Checking Contest (a newly established contest), and the Best Process Mining Dissertation Award.

We would like to thank all those that helped making this event a successful story. First and foremost, Wil van der Aalst and the PADS group, for hosting the conference and managing all the logistics involved. Second, the Program Committee members that helped in the review process. Third, we would like to thank all the authors for their valuable contributions as well as all the academics involved in the committees of the different satellite events, who contributed to create a rich conference program. And last, we would of course like to thank all our sponsors for their support.

Josep Carmona
Mieke Jans
Marcello La Rosa
***Program Co-Chairs ICPM 2019***

# Conference Organization

ICPM 2019

### Organizing Committee

Claudia Graf
Christine Dobbert
Christina Rensinghof
Bas van Zelst
Detlef Wetzeler
Wil van der Aalst

### Inaugural Steering Committee

Wil van der Aalst, *RWTH Aachen, Germany, General Chair*
Josep Carmona, *Universitat Politècnica de Catalunya, Spain, Program Chair*
Mieke Jans, *Hasselt University, Belgium, Program Chair*
Marcello La Rosa, *University of Melbourne, Australia, Program Chair*
Boudewijn Van Dongen, *TU Eindhoven, Netherlands*

### Demo Chairs

Andrea Burattin
Artem Polyvyanyy
Sebastiaan van Zelst

### Process Mining Dissertation Award

Robin Bergenthum
Chiara Di Francescomarino
Marlon Dumas, Chair
Walid Gaaloul
Marcos Sepúlveda
Alessandro Sperduti
Jan Martijn van der Werf

### Doctoral Consortium Chairs

Jan Claes
Boudewijn van Dongen

### Organizers Process Discovery Contest

Josep Carmona
Massimiliano de Leoni
Benoît Depaire

### Chair International Business Process Intelligence Challenge

Boudewijn van Dongen

### Organizers Conformance Checking Challenge

Abel Armas-Cervantes
Luciano Garcia-Banuelos
Jorge Munoz-Gama

# Program Committee
## ICPM 2019

Antonella Guzzo, *University of Calabria, Spain*
Arik Senderovich, *Technion, Israel*
Artem Polyvyanyy, *The University of Melbourne, Australia*
Arthur Ter Hofstede, *Queensland University of Technology, Australia*
Avigdor Gal, *Technion, Israel*
Barbara Weber, *Technical University of Denmark, Denmark*
Benoît Depaire, *Hasselt University, Belgium*
Boudewijn Van Dongen, *Eindhoven University of Technology, Netherlands*
Chiara Di Francescomarino, *Fondazione Bruno Kessler-IRST, Italy*
Chiara Ghidini, *Fondazione Bruno Kessler (FBK), Italy*
Claudio Di Ciccio, *Vienna University of Economics and Business, Austria*
Diogo R. Ferreira, *Universidade de Lisboa, Portugal*
Dirk Fahland, *Eindhoven University of Technology, Netherlands*
Fabrizio Maria Maggi, *Institute of Computer Science - University of Tartu, Estonia*
Felix Mannhardt, *SINTEF Digital, Norway*
Frederik Gailly, *Faculty of Economics and Business Administration, Ghent University, Belgium*
Hajo A. Reijers, *Utrecht University, Netherlands*
Hyerim Bae, *Pusan National University, Korea*
Irina Lomazova, *National Research University Higher School of Economics, Russia*
Jan Claes, *Ghent University, Belgium*
Jan Mendling, *Wirtschaftsuniversität Wien, Austria*
Jan Vanthienen, *Katholieke Universiteit Leuven, Belgium*
Jianmin Wang, *Tsinghua University, China*
Jochen De Weerdt, *Katholieke Universiteit Leuven, Belgium*
Jorge Munoz-Gama, *Pontificia Universidad Católica de Chile, Chile*
Lijie Wen, *Tsinghua University, China*
Luciano García-Bañuelos, *Tecnológico de Monterrey, Mexico*
Manuel Lama Penin, *University of Santiago de Compostela, Spain*
Marco Montali, *KRDB Research Centre, Free University of Bozen-Bolzano, Italy*
Marcos Sepúlveda, *Pontificia Universidad Católica de Chile, Chile*
Marlon Dumas, *University of Tartu, Estonia*
Massimiliano de Leoni, *Department of Mathematics - University of Padua, Italy*
Matthias Weidlich, *Humboldt-Universität zu Berlin, Germany*
Minseok Song, *POSTECH (Pohang University of Science and Technology), Korea*
Moe Wynn, *Queensland University of Technology, Australia*
Nick van Beest, *CSIRO, Australia*
Paolo Ceravolo, *University of Milan, Italy*
Raffaele Conforti, *The University of Melbourne, Australia*
Robin Bergenthum, *Fern Universität in Hagen, Germany*
Sander J.J. Leemans, *Queensland University of Technology, Australia*
Sebastiaan J. van Zelst, *Fraunhofer, Germany*
Seppe Vanden Broucke, *Katholieke Universiteit Leuven, Belgium*
Suriadi Suriadi, *Queensland University of Technology, Australia*
Thomas Chatain, *LSV, ENS Paris-Saclay, Cachan, France*
Xixi Lu, *Utrecht University, Netherlands*

# Reviewers
ICPM 2019

### Additional Reviewers

Jaehun Park, *Daegu Haany University, Korea*
Gert Janssenswillen. *Hasselt University, Belgium*
Roee Shraga, *Technion, Israel*
Mathilde Boltenhagen, *Chatain, LSV, France*
Eric Verbeek, *Eindhoven University of Technology, Netherlands*
Bernardo Yahya, *Hankuk University of Foreign Studies, Korea*

# PRETSA: Event Log Sanitization for Privacy-aware Process Discovery

Stephan A. Fahrenkrog-Petersen, Han van der Aa, Matthias Weidlich
*Humboldt-Universität zu Berlin, Berlin, Germany*
*Email: {stephan.fahrenkrog-petersen, han.van.der.aa, matthias.weidlich}@hu-berlin.de*

***Abstract*—Event logs that originate from information systems enable comprehensive analysis of business processes, e.g., by process model discovery. However, logs potentially contain sensitive information about individual employees involved in process execution that are only partially hidden by an obfuscation of the event data. In this paper, we therefore address the risk of privacy-disclosure attacks on event logs with pseudonymized employee information. To this end, we introduce PRETSA, a novel algorithm for event log sanitization that provides privacy guarantees in terms of k-anonymity and t-closeness. It thereby avoids disclosure of employee identities, their membership in the event log, and their characterization based on sensitive attributes, such as performance information. Through stepwise transformations of a prefix-tree representation of an event log, we maintain its high utility for discovery of a performance-annotated process model. Experiments with real-world data demonstrate that sanitization with PRETSA yields event logs of higher utility compared to methods that exploit frequency-based filtering, while providing the same privacy guarantees.**

## 1. Introduction

Event logs that are recorded by information systems are the starting point for process mining, i.e., the data-driven analysis of qualitative and quantitative properties of business processes [1]. Specifically, discovery algorithms construct a process model from an event log, thereby formalizing the recorded execution dependencies between a process' activities [2]. Focusing on quantitative properties, such a model is annotated with performance information to facilitate process simulation [3], e.g., to assess the average cycle time.

With the potential of process mining unfolding, organizations intensify their efforts for accurate and fine-granular recording of their processes. Once a process involves manual processing, however, the resulting event logs enable sensitive conclusions on individual employees. As such, event logs may breach privacy [4], violating informal self-determination, i.e., an individual's ability to control, who has access to their personal data [5]. Avoidance of potential privacy breaches is not only an ethical consideration, but may be enforced by legislation. An example is the European General Data Protection Regulation (GDPR) that prohibits processing of personal data unless necessary for a specific purpose [6].

Privacy-aware processing of event logs may be approached based on methods for information security that prevent unauthorized access, use, and disclosure of data [4]. Specifically, data confidentiality may be achieved by restricting the access and interpretation of data by pseudonymization, i.e., the obfuscation of data to prevent direct identification of entities, or anonymization, i.e., the permanent de-identification of data that renders conclusions on entities impossible even when relying on additional data [7].

Such methods, however, are insufficient in terms of privacy in some application contexts as they do not prevent frequency-based attacks [8]. For instance, pseudonymization of employee information in a log does not achieve privacy, if it is known that only a certain employee may execute a specific activity. A different angle therefore is the sanitization of data to give well-defined privacy guarantees. Examples for such guarantees are $k$-anonymity [9] (each entity cannot be distinguished from at least $k$ other entities) or differential privacy (adding noise to the data, there is a guaranteed level of ambiguity trying to reconstruct the original data) [10].

Many notions of privacy have a parameter that enables fine-tuning of the strengths of the given guarantee, e.g., $k$ in $k$-anonymity and a privacy budget $\epsilon$ in differential privacy. Typically, there is a trade-off between the strengths of a privacy guarantee induced by data sanitization and the loss in utility of the data for some analysis question. For instance, aggregating the events of several instances of a process yields stronger $k$-anonymity (a higher $k$ value), but perturbs the execution dependencies recorded between activities. Adding more noise to an event log (smaller privacy budget $\epsilon$), yields higher ambiguity in differential privacy, but also decreases the accuracy of process simulation. The actual loss in analysis utility incurred by privacy guarantees depends on the exact definition of respective data sanitization, though. This raises the question of *how to sanitize data such that data utility is maximized under a given privacy guarantee?*

In this paper, we address the above question for a specific application context in process mining. This context is defined by an attack model, which determines the privacy guarantees to consider, and a type of process analysis, which determines how to assess the utility of a sanitized event log.

We consider a *trace linking attack* on an event log with pseudonymized employee information that correlates events with background knowledge on resources. This attack involves (i) identity disclosure, whether an event is related to an employee; (ii) membership disclosure, whether events of an employee are contained in the log; and (iii) attribute disclosure, whether an employee can be characterized through attribute values of events. We aim to prevent this attack by

978-1-7281-0919-0/19/$31.00 ©2019 IEEE
DOI 10.1109/ICPM.2019.00012

sanitizing an event log before it is used to discover a process model, including annotations of activity durations. As such, the utility of the sanitized log is assessed in terms of the change of the model discovered from the sanitized log in comparison to the one discovered from the original log.

For this setting, we present *PREfix-Tree based event log SAnitisation for t-closeness*, shortly *PRETSA*, an event log sanitization algorithm that prevents membership and identity disclosure by k-anonymity and protects against attribute disclosure by t-closeness. In essence, PRETSA constructs a prefix tree representation of an event log that is annotated with frequencies and attribute values. This tree is then stepwise transformed by relocating and merging sub-trees until the required privacy guarantees have been obtained. This way, transformations of the event log are comparatively fine-granular, which implies a modest loss in the log's utility.

We evaluate PRETSA against a baseline that achieves the respective privacy guarantees by filtering the event log. Our experiments with three real-world datasets indicate that the event logs obtained using PRETSA have a high utility for both process discovery and performance analysis. Furthermore, we show that PRETSA outperforms the baseline along various evaluation dimensions and yields good results, even when the baseline fails to provide any results at all.

In the remainder of the paper, we provide a motivating example (Section 2), before formalizing the considered attack model and privacy guarantees (Section 3). We then introduce PRETSA, our algorithm for event log sanitisation (Section 4) and present an empirical evaluation (Section 5). Finally, we review related work (Section 6) and conclude (Section 7).

## 2. Motivation

To motivate the need for event log sanitization, consider an order handling process, which encompasses activities related to the creation of purchase orders (PO) (*create_po*), updating them (*update_po*), receiving goods (*receive_gd*), and checking, paying, and rejecting invoices (*check_in*, *pay_in*, *reject_in*). Assume that events are recorded for this process, which enables process discovery and performance analysis. While an event log that contains the recorded event data is required to apply these techniques, it is important to recognize that the previously discussed ethical and legal considerations prevent the manager from collecting or disclosing data that compromise the identity of individual employees.

Straightforwardly, this means that an event log must not contain information about which employee performed which events. However, for someone with malicious intent, i.e., an *attacker*, information on the sequencing of events may be enough to relate employees to the execution of certain events [11]. This, particularly, holds if an attacker possess organizational knowledge (e.g., a manager). For instance, a manager may be aware that, for POs that have been updated, only four employees are allowed to check the corresponding invoice. By combining such background knowledge with the traces in an event log, adversaries could derive sensitive information, such as:

- That an event was performed by a specific employee (*identity disclosure*).
- That the data of a specific employee is included in the event log (*membership disclosure*).
- That an employee can be characterized by execution-related data, e.g. performance data (*attribute disclosure*)

For example, consider a scenario in which some POs have been updated after goods receipt. If an adversary knows that Sue is one of the few employees that are allowed to subsequently check the corresponding invoice, the adversary would be able to identify the specific events that were performed by Sue (identity closure) with high accuracy.

To reduce the probability that such an attack will succeed, event logs must be sanitized to protect the privacy of an organization's employees. One way to achieve this is to alter event logs so that they meet privacy guarantees. An example for such a guarantee is *k-anonymity*, which bars the disclosure of infrequently occurring process behavior. From a process mining perspective, a downside of such a guarantee is that information may become obscured by sanitization.

TABLE 1: Exemplary sequences of activity executions.

| | Sequence variant | # |
|---|---|---|
| $\sigma_1$ | *create_po*,*update_po*,*receive_gd*,*check_in*,*pay_in* | 10 |
| $\sigma_2$ | *create_po*,*update_po*,*receive_gd*,*check_in*,*reject_in* | 5 |
| $\sigma_3$ | *create_po*,*receive_gd*,*update_po*,*check_in*,*pay_in* | 7 |
| $\sigma_4$ | *create_po*,*receive_gd*,*update_po*,*check_in*,*reject_in* | 5 |
| $\sigma_5$ | *create_po*,*receive_gd*,*update_po*,*update_po*,*check_in*,*pay_in* | 1 |

Assume that an event log contains events that represent sequences of activity executions as detailed in Table 1. A straightforward manner to ensure k-anonymity, would be to remove any variant from the log that occurs less than $k$ times. Given that only one variant occurs at least eight times, a requirement for $k$-anonymity with $k = 8$, would yield a sanitized event log that contains only 10 sequences of events that all represent variant $\sigma_1$. While this log indeed provides the desired privacy guarantee, it also hides a considerable amount of information on the presence and frequency of other sequence variants. When applying process discovery techniques on this sanitized log, we would therefore discover a process model that only captures a fraction of the actually recorded process behavior.

## 3. Event Log Privacy

Next, we provide a formal model of the illustrated setting, including a model of event logs (Section 3.1), the considered trace linking attack (Section 3.2), and privacy guarantees to cope with the attack (Section 3.3).

### 3.1. Event Log Model

To formalize privacy requirements for event logs, we adopt an event model that builds upon a set of activity identifiers $\mathcal{A}$ (activities, for short) and a set of resources $\mathcal{R}$ (i.e., employees). For the execution of an activity by a resource, we further consider execution-related data. Here,

we incorporate this data as a sensitive attribute of a domain $\mathcal{S}$ that is relevant for the analysis. Note that execution-related data that is not important for process analysis does not pose any privacy challenge, as it may simply be discarded.

Let $\mathcal{I}$ be a set of event identifiers. Then, by $\mathcal{E} = \mathcal{I} \times \mathcal{A} \times \mathcal{R} \times \mathcal{S}$, we denote the universe of events that may be recorded by an information system. As such, each recorded *event* $e = (i, a, r, s) \in \mathcal{E}$ represents an execution of an activity $a$ by a resource $r$ with $s$ as the sensitive attribute value, whereas $i$ is a unique identifier, i.e., for $(i, a, r, s), (i', a', r', s') \in \mathcal{E}$ it holds that $i = i'$ implies that $(i, a, r, s) = (i', a', r', s')$.

A single execution of a process, called a *trace*, is a finite sequence of events $t = \langle e_1, \ldots, e_n \rangle \in \mathcal{E}^*$. By $\mathcal{T}$, we denote the universe of all traces. We define an *event log* as a set of traces, $L = \{t_1, \ldots, t_m\}$ with $t_j \in \mathcal{T}$ for $1 \leq j \leq m$.

### 3.2. Attack Model

We consider a scenario in which a performance-annotated process model shall be discovered from an event log, where performance information is given by the sensitive attribute. Since an event log, as defined above, contains information on individual resources, i.e., the employees that executed a certain activity, it cannot be disclosed. Rather, data that is not relevant for the intended analysis shall be projected. We therefore consider a projection $\pi : \mathcal{E} \to \mathcal{I} \times \mathcal{A} \times \mathcal{S}$ with $\pi(i, a, r, s) = (i, a, s)$ that removes information on the resource from an event. This projection is lifted to a trace and a log, respectively, by applying it to all contained events, i.e., $\pi(t) = \langle \pi(e_1), \ldots, \pi(e_n) \rangle$ and $\pi(L) = \{\pi(t_1), \ldots, \pi(t_l)\}$.

The projected log $\pi(L)$ does not allow for direct conclusions on which activity execution (i.e. which event) was performed by whom (*identity disclosure*), whether events of a resource are part of the log (*membership disclosure*), or on a characterization of resources by values of the sensitive attribute (*attribute disclosure*).

The above information may be revealed, though, if the projected log is combined with background information. Here, we consider such information on the relation between resources and activities. In practice, such information is frequently available: It may stem, for example, from the definition of organizational responsibilities (resources differ in the sets of activities that they are obliged to execute) [12]; role-based access control in information systems (resources differ in the sets of activities that they are allowed to execute) [13]; or information on shift schedules (resources differ in their availability to execute certain sets of activities) [14].

In this work, we incorporate a rather expressive notion of background information that does not only provide insights into possible assignments of activities to resources, but potentially limits these assignments based on activity patterns. This enables us to model application contexts in which the assignment of activities to resources is controlled in a fine-granular manner. As an example, reconsider the invoice handling process of Section 2. In this process, the payment of an invoice may only be performed by specific employees, if the purchase order was previously updated after goods receipt, which indicates an abnormal process execution. We formalize such background information by a function $b : \mathcal{A}^* \times \mathcal{A} \to 2^{\mathcal{R}}$, so that $b(\langle a_1, \ldots, a_n \rangle, a) = \{r_1, \ldots, r_m\}$ captures that an activity $a$ in a trace in which activities $a_1, \ldots, a_n$ have been executed already in the respective order, may be assigned to one of the resources $\{r_1, \ldots, r_m\}$. For example, $b(\langle receive\_gd, update\_po \rangle, pay\_inv) = \{Per, Sue, Amy, Jim\}$ models that *Per*, *Sue*, *Amy*, and *Jim* may pay an invoice of a PO that was updated after goods receipt.

Using such background knowledge, we consider an attack on a projected event log as a specific type of sequence linking attack [11]. That is, given a projected log $\pi(L)$, a *trace linking attack* is an attempt for:

- *Identity disclosure*: Identify a function $work : \mathcal{R} \nrightarrow \mathcal{I} \times \mathcal{A} \times \mathcal{S}$ that assigns an event $(i, a, s)$ of a projected trace $t' \in \pi(L)$ to resource $r$, such that $(i, a, r, s)$ is an event of the original trace $t \in L$, i.e., $t' = \pi(t)$.
- *Membership disclosure*: For a resource $r \in \mathcal{R}$, identify whether there exists a projected trace $t' \in \pi(L)$, such that the original trace $t \in L$, $t' = \pi(t)$, contains an event $(i, a, r, s)$ for some $i \in \mathcal{I}$, $a \in \mathcal{A}$, and $s \in \mathcal{S}$.
- *Attribute disclosure*: Given a distance function over two bags of values of the sensitive attribute, $d : \mathcal{B}(\mathcal{S}) \times \mathcal{B}(\mathcal{S}) \to \mathbb{R}$, for an activity $a \in \mathcal{A}$, identify whether the distribution of values of events related to activity $a$ differs by at least $\delta \in \mathbb{R}$ for some resource $r \in \mathcal{R}$, i.e., $d(\sum_{r \in \mathcal{R}} \Gamma(r, a), \Gamma(r, a)) > \delta$ with $\Gamma(r, a) = [s \mid t = \langle e_1, \ldots, e_n \rangle \in L, 1 \leq j \leq n : e_j = (i, a, r, s)]$.

The above attack is facilitated using background information, as follows. For a projected log $\pi(L)$, the background information $b$ induces equivalence classes: Each activity pattern of an element $(\langle a_1, \ldots, a_n \rangle, a)$ in the domain of $b$ defines a class that contains a projected trace $\langle (i'_1, a'_1, s'_1), \ldots, (i'_m, a'_m, s'_m) \rangle \in \pi(L)$, if the trace shows the pattern, i.e., there exists a mapping $\lambda : \{1, \ldots, n\} \to \{1, \ldots, m\}$ such that $a_j = a'_{\lambda(j)}$ for $1 \leq j \leq n$ and $\lambda(j) < \lambda(j+1)$ for $1 \leq j < n$. In the worst case, each prefix of activities $\langle a'_1, \ldots, a'_m \rangle$ induces an equivalence class.

For each equivalence class, the background information then defines for each activity, the set of resources that may have been involved in the respective events. Taking up the above example (Table 1) and the aforementioned background information, we derive an equivalence class that contains eight traces, i.e., all traces that correspond to variants $\sigma_3$ and $\sigma_5$. The background information reveals that $\{Per, Sue, Amy, Jim\}$ could have executed the payment of the invoice.

A trace linking attack discloses identity, membership, or attribute values for each of the equivalence classes. We illustrate this for identity disclosure for the above example. Assume that a resource could have paid at most four invoices, i.e., at most four events in the equivalence class can be related to any resource. Picking one resource, say *Sue*, identity disclosure is the construction of $work(Sue)$, assigning events of the projected log to *Sue*. The maximal probability being successful in this construction is bounded by the ratio of the number of events of an activity that may be assigned to a resource (four) and the number of events in the equivalence class (eight). Thus, the maximal probability of correctly assigning events for invoice payments to *Sue* is $4/8 = 0.5$.

### 3.3. Privacy Guarantees

To reduce the risk that an adversary performs a trace linking attack successfully, the projected event log shall have characteristics that lower the probability of disclosure. This may be achieved for identity and membership disclosure by adopting the notion of *k-anonymity* [9], defined as follows.

**Definition 1.** *(k-anonymity) Let $\pi(L)$ be a projected event log. $\pi(L)$ satisfies k-anonymity, if and only if each equivalence class of $\pi(L)$ contains at least k events.*

As discussed above, the equivalence classes of $\pi(L)$ are induced by the background information $b$ in our model: Each activity pattern of an element of the domain of $b$ defines one equivalence class. By requiring that each class contains at least $k$ events, k-anonymity gives us a direct bound on the maximal probability that disclosure succeeds.

Let $\tau(a)$ be the maximal number of events that can represent the execution of an activity $a \in \mathcal{A}$ by any resource. Then, the maximal probability of successful identity disclosure for an event $e = (i, a, s)$ of a projected trace is bounded by $P(e \in work(r)) \leq \tau(a)/k$. Put differently, guessing the correct assignment of an event $e = (i, a, s)$ to a resource ($e \in work(r)$) in the equivalence class induced by the background information will succeed with a probability of at most $\tau(a)/k$ under k-anonymity.

Taking up the above example, the discussed equivalence class contains eight traces, which corresponds to 8-anonymity when considering only the invoice payment activity and yields a bound of $4/8 = 0.5$. However, if the projected log would satisfy, e.g., 16-anonymity, we would get a tighter bound for the probability of successful disclosure: The maximal probability of correctly assigning events related to the invoice payments to *Sue* would drop to $4/16 = 0.25$.

The above notion of k-anonymity also helps to guard against membership disclosure. Here, we consider the disclosure to be successful if it can be decided with certainty that an event representing an activity execution of a resource is part of the projected log. However, we see that this is not possible if each equivalence class contains at least $k$ events, so that $k > \tau(a)$ for all activities $a \in \mathcal{A}$. In that case, the events in each equivalence class relate to at least two resources, which renders it impossible to conclude with certainty on the association of an event to a specific resource.

Ensuring that the projected log satisfies k-anonymity does not protect against attribute disclosure, though. The values of the sensitive attribute of the events of an activity in an equivalence class may show a distribution that is very different from one over all events of the respective activity, thereby enabling conclusions on the identity of the resources linked to the events in the equivalence class.

To guard against such disclosure, we adopt an enhancement of k-anonymity, called *t-closeness* [15], which limits the amount of information an adversary can gain through the sensitive attribute. Based on [15], with $d : \mathcal{B}(\mathcal{S}) \times \mathcal{B}(\mathcal{S}) \to \mathbb{R}$ as a distance function (e.g., the Earth Mover's distance [16]), we define this notion as follows:

**Definition 2.** *(t-closeness) Let $\pi(L)$ be a projected event log and $d$ a distance function. An equivalence class of $\pi(L)$ has t-closeness, if for all activities $a \in \mathcal{A}$, the difference in the value distribution over all events for $a$ in $\pi(L)$, and those in the equivalence class is less or equal than a threshold $t \in \mathbb{R}$, i.e., $d(\Omega(a), \Omega'(a)) \leq t$ with $\Omega(a) = [s \mid t = \langle e_1, \ldots, e_n \rangle \in \pi(L), 1 \leq j \leq n : e_j = (i, a, s)]$ and $\Omega'(a)$ as the restriction of $\Omega(a)$ to events of the equivalence class. $\pi(L)$ has t-closeness, if all its equivalence classes have t-closeness.*

The notion of t-closeness helps to prevent attribute disclosure, by requiring that none of the equivalence classes induced by background information differs significantly, as defined by parameter $t$, in terms of the value distribution of the sensitive attribute. As such, it prevents any conclusion from the equivalence class on the resources involved in its events through the values of the sensitive attribute.

If a projected event log $\pi(L)$ fulfills *k-anonymity* and *t-closeness*, it is guarded against the trace linking attack by the discussed privacy guarantees. If that is not the case, however, the log needs to be *sanitized*. In this work, we consider such a sanitization step as the application of a function $f$ to the event log, which is parametrized by $k$ and $t$ for the desired strengths of the privacy guarantees. It shall yield an event log, $f(\pi(L), k, t) = \pi(L)'$ that adheres to the required privacy guarantees, while preserving utility for process mining.

## 4. The PRETSA Algorithm

In this section, we introduce the *PRETSA*, an algorithm for *PREfix-Tree based event log SAnitization for t-closeness*. It provides an implementation of the aforementioned sanitization function $f$ for a projected event log. PRETSA is inspired by the BF-P2kA-algorithm (Brute Force Pattern-Preserving k-Anonymization) presented in [11] to sanitize personal data of sequential nature, such as sequences of visited locations. Below, we simplify notation and discuss sanitization of an event log $L$ into a log $L'$, since the projection of resource information ($\pi(L)$ and $\pi(L)'$) is an orthogonal aspect.

With PRETSA, we aim to maximize the utility of a sanitized event log $L'$ for process discovery and process model enhancement. Process discover techniques generally derive *directly follows graphs* (DFG), cf. [17], [18], [19], or similar relations from event logs. In this context, DFGs capture which activities directly succeed each other in an event log's traces and with what frequency. To maximize the utility preserved in a sanitized log $L'$, the DFG that can be constructed from $L'$ should resemble the DFG of the original log $L$. Therefore, we strive to retain as many of entries of the directly follows relations. In addition, we strive to preserve the quality of activity annotations, used for process model enhancement, by reducing the impact of sanitization on the distribution of values for the sensitive attribute.

To preserve utility, PRETSA transforms an event log into a prefix tree [20]. In a prefix tree, each node is a set of events that all represent executions of the same activity and share the same prefix of activity executions in their respective traces. We capture information about a node as a

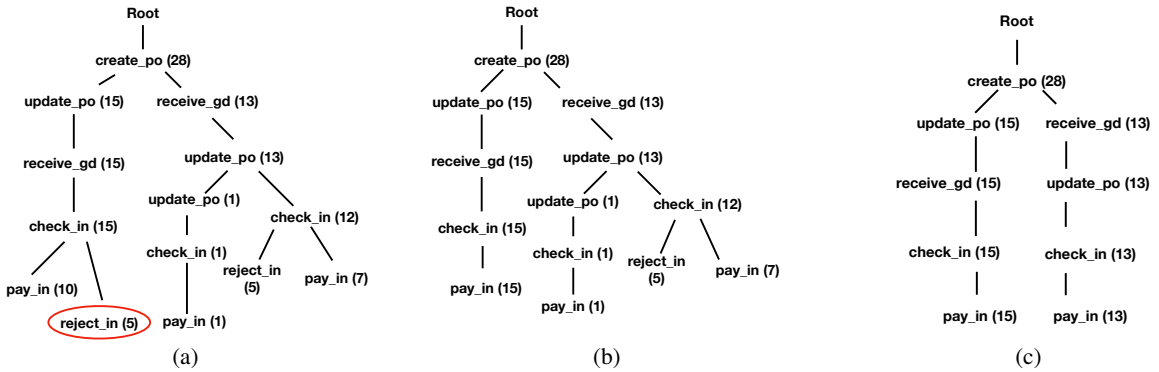



Figure 1: (a) Prefix tree of the example; (b) tree obtained after pruning the highlighted node and reconstructing the tree; (c) final tree returned by PRETSA for the example, setting $k = 8$ and $t = 1.0$.

tuple $n = (a, T, S)$, where $a \in \mathcal{A}$ is the activity of the events; $T \subseteq \mathcal{T}$ is the set of traces that contain the events; $S \in \mathcal{B}(\mathcal{S})$ is the bag of values of the sensitive attribute.

To illustrate the notion of a prefix tree, Figure 1a depicts the tree corresponding to sequences of activity executions in Table 1. For example, the tree shows that the log contains 15 traces that start with the sequence *create_po*,*update_po*,*receive_gd*,*check_in*, that 10 of those are followed by event related to *pay_in*, whereas the remaining five are followed by an event representing activity *reject_in*.

From a privacy perspective, each node in a prefix tree corresponds to a potential equivalence class that may be induced by some background information. In fact, it considers the worst case, in which each prefix of a sequence of executed activities observed in the log defines a separate equivalence class. To fulfill $k$-anonymity and $t$-closeness, PRETSA manipulates the prefix tree, such that each node fulfills these properties. The general approach is defined in Alg. 1. We illustrate the algorithm in Figure 1 for the privacy parameters $k = 8$ and $t = 1.0$. The main steps of the algorithm are as follows:

**Check privacy guarantees:** The algorithm traverses the prefix tree $P$ in a depth-first manner (line 4), until it reaches a node $n = (a, T, S)$ that violates a privacy guarantee (line 5):

- $k$-anonymity is violated, if $\mid T \mid < k$, i.e., the events of $n$ originate from less than $k$ traces (see Definition 1).
- $t$-closeness is violated, if $d(\Omega(a), S) > t$ with $\Omega(a)$ as the bag values of the sensitive attribute of all events of activity $a$ of all traces in the log, i.e., if the distance between the overall value distribution and the one at the node is greater than the threshold $t$ (see Definition 2).

For the example in Figure 1a, PRETSA identifies a first violation for the highlighted node. This node violates $k$-anonymity, given that $\mid T \mid = 5 < k = 8$.

**Tree update:** If a violation is detected for node $n = (a, T, S)$, we disassociate the traces $T$ from all of $n$'s ancestors (line 6). For instance, pruning the highlighted node in Figure 1a, we remove the five traces that represent the execution sequence *create_po*,*update_po*,*receive_gd*,*check_in*,*reject_in* from the four nodes on the path from the root to the highlighted node. Then, we remove the node and its descendants (if any) from $P$ (line 7).

**Algorithm 1** PRETSA(L,k,t)

**INPUT:** A event log $L$, the parameter $k$, a threshold $t$
**OUTPUT:** A event log $L'$

```
1:  P ←constructPrefixTree(L)
2:  repeat
3:      hasChanges ← false
4:      for all n ∈ DFS(P) do
5:          if violatesPrivacyGuarantees(t, k, n) then
6:              updateAncestors(P, n)
7:              prune(P, n)
8:              t' ← findMostSimilar(P, n.traces)
9:              P ← reconstructTree(P, t')
10:             hasChanges ← true
11: until ¬ hasChanges
12: return generateEventLog(P)
```

**Find most similar remaining traces:** Next, for each trace $t \in T$ of the pruned node $n$, PRETSA identifies a trace $t'$ that is most similar (line 8). Here, we consider similarity in terms of the edit (i.e., Levenshtein) distance [21]. For instance, for the highlighted node, the path representing the sequence of activities *create_po*,*update_po*,*receive_gd*,*check_in*,*pay_in*, leading to the far-left leaf node, is most similar (edit distance of one).

**Reconstruct tree:** Each trace $t \in T$ of the pruned node $n = (a, T, S)$ is then incorporated into all nodes $n' = (a', T', S')$, where $T'$ contains the selected, most similar trace $t'$, which involves adding the events once their activity $a$ has been set to $a'$ (line 9). For all events that have been transformed by PRETSA, the respective values of the sensitive attribute are discarded and replaced by a random value, which is drawn from the distribution of $\Omega(a)$ for events of activity $a$. For instance, after incorporating the highlighted node into the far-left leaf node, the reconstructed prefix tree will contain 15 traces for the sequence of activity executions *create_po*,*update_po*,*receive_gd*,*check_in*,*pay_in*, rather than the original 10 traces, as shown in Figure 1b.

**Termination:** The algorithm transforms the prefix tree iteratively, until it is fully traversed without identifying a single violation (line 11). For the running example, the final

tree is shown in Figure 1c. Based on the obtained prefix tree, PRETSA returns a sanitized event log as the set of all traces represented by the tree (line 12).

Applying PRETSA to our example, with $k = 8$ and $t = 1.0$, we end up with the sanitized event log that contains traces of the following sequences of activity executions:

| | |
|---|---|
| *create_po,update_po,receive_gd,check_in,pay_in* | 15 |
| *create_po,receive_gd,update_po,check_in,pay_in* | 13 |

A naive approach that deletes all traces that violate the privacy requirements would yield a less representative log:

| | |
|---|---|
| *create_po,update_po,receive_gd,check_in,pay_in* | 10 |

## 5. Evaluation

This section presents an experimental evaluation of the PRETSA algorithm. We show that PRETSA enables the sanitization of event logs, while preserving utility for process mining. Section 5.1 introduces the three real-world event logs used in our experiments, Section 5.2 describes the experimental setup, while Section 5.3 discusses the results.

### 5.1. Datasets

We conducted our evaluation experiments based on the three real-world event logs characterized in Table 2. The table shows that the characteristics of the event logs differ considerably. Most notably the number of cases per variant ranges from an average of 1.2 to 651.0. Given that this factor is crucial to the performance of a sanitization approach, we believe that the utilized data collection is well-suited to achieve a high external validity of the results.

TABLE 2: Characteristics of the utilized event logs

| Name | Cases | Variants | Cases per variant Avg. | Cases per variant Max. |
|---|---|---|---|---|
| Traffic Fines [22] | 150,370 | 231 | 651.0 | 56,482 |
| CoSeLog [23] | 1,434 | 116 | 12.4 | 713 |
| Sepsis [24] | 1,050 | 846 | 1.2 | 35 |

### 5.2. Experimental Setup

To conduct our experiments, we implemented PRETSA as a stand-alone Python program that is available on Github[1]. We use this implementation to analyze two aspects:

**1) Impact of k-anonymity on discovered models.** Guaranteeing $k$-anonymity can affect the quality of process models discovered from sanitized event logs. To assess this, we first discover a process model from a sanitized event log $L'$ using the Inductive Miner infrequent [19]. Afterwards, we quantify the quality of this discovered model by determining its *fitness* [25] and *precision* [26] with respect to the original event log $L$. In this way, we show how accurately the discovered process models discovered capture the behavior of the original non-sanitized log. In our experiments, we assess the impact of various $k$ values, using $k = \{2, 4, 8, 16, 32, 64, 128, 256\}$.

**2) Impact of t-closeness on model enhancement.** Guaranteeing $t$-closeness can affect the accuracy of process model annotations derived from sanitized event logs. To determine this impact, we compare annotations, in particular execution times, derived from an event log $L'$, sanitized by PRETSA, to annotations derived from the original event log $L$. If an event log did not contain an end timestamp for an activity, we generate the execution time of an event $e_i$ as the time difference between the start time of the event $e_i$ and the start time of the following event $e_{i+1}$. In our experiments, we assess the impact of various $k$ values, using $t = \{0.025, 0.050, 0, 075, 0.100\}$.

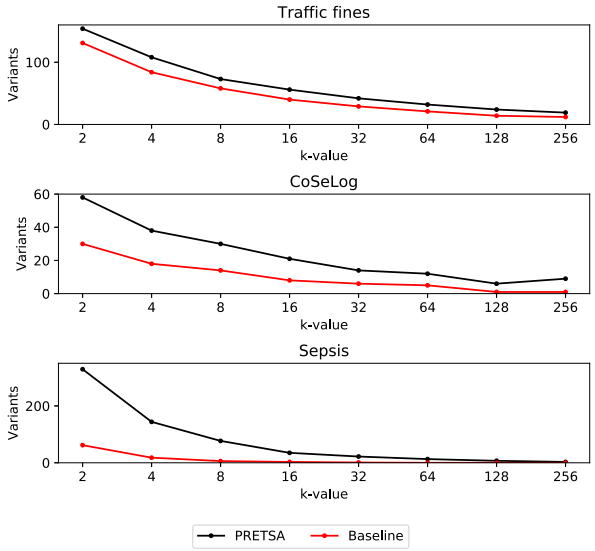



Figure 2: Number of variants retained for $k$-anonymity.

We compare the results obtained using PRETSA to a baseline approach. For this baseline, we provide privacy guarantees in the following, rudimentary manner:

**k-anonymity baseline.** To guarantee $k$-anonymity for the baseline log $L'_{BL}$, we remove traces that represent an activity sequence that occurs less than $k$-times in $L$.

**t-closeness baseline.** To guarantee $t$-closeness for the baseline log $L'_{BL}$, we compare the distribution of the execution times for an activity $a \in \mathcal{A}$ when it is part of traces of a specific sequence variant to the overall distribution of the throughput times for activity $a$ in the original log $L$. If these two distributions statistically differ, then the $t$-closeness requirement is not met. We then discard all traces of the respective sequence variant in the construction of log $L'_{BL}$.

### 5.3. Results

**Impact of $k$-anonymity on discovered models.** Figure 2 shows how the contents of event logs are preserved for varying $k$ values. We see that, for both PRETSA and the

1. https://github.com/samadeusfp/PRETSA

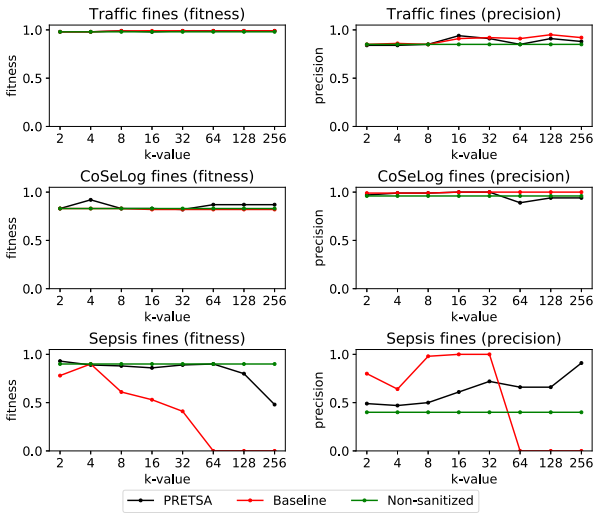



Figure 3: Fitness and precision results based on the Inductive Miner infrequent (default parameters).

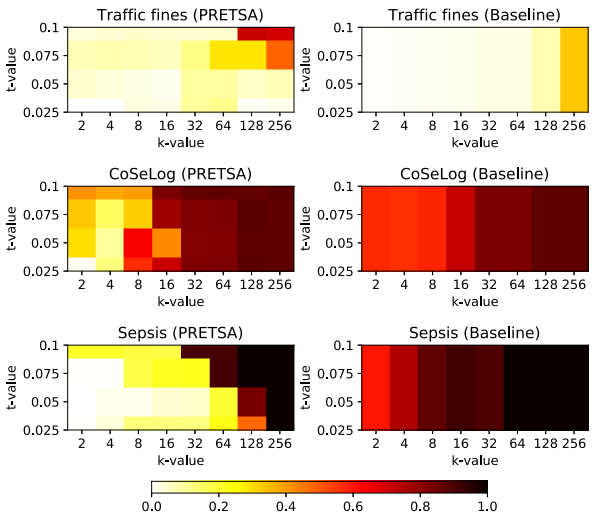



Figure 4: Distance between annotations in sanitized versus non-sanitized event logs (with 0.0 as equal annotations).

baseline, the number of contained variants considerably decreases for higher values of $k$. However, using PRETSA, we are able to consistently retain more variants than the baseline. For instance, with $k = 4$, log $L'_{BL}$ contains just 18 variants for the Sepsis case, whereas the log obtained by PRETSA still contains 144 variants. For this case, with $k \geq 64$ the baseline log actually contains zero variants, because the most common trace occurs only 35 times. PRETSA on the other hand always retains at least 3 variants.

Similar trends can be observed in Figure 3, in which we depict the log fitness and precision obtained when comparing an original event log $L$ to the process models discovered by applying the Inductive Miner on the sanitized event logs. This figure also depicts the results obtained when discovering models based on non-sanitized event logs.

For the *traffic fines* case, the figure shows that the fitness and precision values do not differ considerably across the three approaches. This outcome is in line with the expectations for this case, because most variants in the log are rather common. As a result, even a naive sanitization approach, i.e., the baseline, has a limited impact on the results. For the *CoSeLog* case, we observe that PRETSA outperforms the baseline for most $k$-values, in terms of log fitness, but may lead to smaller precision values. The biggest differences among the approaches can be observed for the *Sepsis* case, because the baseline discards a considerable number of variants. For instance, for $k = 64$, PRETSA still achieves a fitness of 0.90, whereas the baseline yields a null result. This demonstrates that PRETSA is particularly useful for less-structured processes.

**Impact of $t$-closeness on model enhancement.** Figure 4 presents heatmaps that depict the impact of sanitization for $t$-closeness on the accuracy of execution time annotations. Given specific $k$ and $t$ values, the figure indicates how close the average execution times of activities in the sanitized log $L'$ are to the original average in the original logs. In particular, we represent the distance between the two averages, where a distance of 0.0 means that they are equal. Overall, PRETSA is shown to have a higher annotation accuracy than the baseline. However, the results across the three cases differ.

For the *traffic fines* case, we observe that both PRETSA and the baseline provide accurate annotations. As noted earlier, this is due to the high number of traces per sequence variant in this log. Consequently, privacy guarantees can be provided with limited impact on utility. For the *CoSeLog* case, there are considerable differences between the original annotations and the annotations derived from sanitized event logs. For $k > 8$, the accuracy drops, as an increasing number of events are omitted, although PRETSA still achieves better accuracy than the baseline. These observations are interesting, because we previously showed that the impact of $k$-anonymity on the quality of discovered process models is rather limited for this case. Finally, for the *Sepsis* case, we observe clear differences between the performance of PRETSA and the baseline. For $k \leq 32$ , the annotations of PRETSA are highly accurate. By contrast, the baseline approach struggles to provide accurate annotations for any $k$ and $t$.

Overall, our evaluation experiments show that PRETSA outperforms the rudimentary baseline methods in providing privacy guarantees in most evaluation settings. However, the results also illustrate that strong privacy requirements impact the quality of results obtained by process model discovery and enhancement techniques.

## 6. Related Work

The presented work relates to the consideration of privacy in the contexts of process and sequence mining.

The importance of privacy has been widely recognized in the area of information systems engineering, cf., [27], [28]. Recently, privacy concerns have been acknowledged in the specific context of process mining, for instance by Van der Aalst [29]. Mannhardt et al. [4] provide an overview of privacy challenges and guidelines in this regard, covering

both technological as well as organizational aspects. Work by Rafiei et al. [7] aims to provide confidentiality in process mining through encryption of event log information. However, this approach does not provide any privacy guarantees.

In the context of sequence mining, Monreal et al. [11] provide an approach that achieves $k$-anonymity for sequential data, which served as basis for the PRETSA algorithm proposed in our work. Crucially, the work by Monreal et al. does not consider data payload, which is required in order to prevent attribute disclose attacks, such as achieved by PRETSA through $t$-closeness guarantees. Moreover, the iterative, step-wise nature of PRETSA allows it to preserve more event log information.

## 7. Conclusion

In this paper, we considered how to provide privacy guarantees for events logs while preserving data utility for process mining analyzes. For this purpose, we introduced a model for privacy-disclosure attacks that utilizes execution sequences to derive sensitive information. To counter such attacks, we developed the PRETSA algorithm to sanitize event logs while providing privacy guarantees in terms of $k$-anonymity and $t$-closeness. PRETSA preserves utility for process mining through step-wise transformation of the prefix-tree representation of an event log. An experimental evaluation using three real-world data sets demonstrates that PRETSA indeed achieves this goal: the evaluation results show that sanitization with PRETSA yields event logs of higher utility compared to a baseline using frequency-based filtering, while providing the same privacy guarantees.

In future research, PRETSA may be extended so that it avoids reaching local optima and instead converges into a global optimum, in order to preserve more event log information. Moreover, we strive to develop sanitization techniques that provide even stronger privacy guarantees, in the form of *differential privacy* [10].

**Acknowledgments.** Parts of this research were funded by the Alexander von Humboldt Foundation.

## References

[1] W. M. Van der Aalst, *Process mining: data science in action*. Springer, 2016.

[2] A. Augusto, R. Conforti, M. Dumas, M. La Rosa, F. M. Maggi, A. Marrella, M. Mecella, and A. Soo, "Automated discovery of process models from event logs: Review and benchmark," *TKDE*, in print, 2018.

[3] W. M. P. van der Aalst, "Business process simulation survival guide," in *Handbook on BPM 1*. Springer, 2015, pp. 337–370.

[4] F. Mannhardt, S. A. Petersen, and M. F. Oliveira, "Privacy challenges for process mining in human-centered industrial environments," in *IE*. IEEE, 2018, pp. 64–71.

[5] T. Asikis and E. Pournaras, "Optimization of privacy-utility trade-offs under informational self-determination," *CoRR*, vol. abs/1710.03186, 2017.

[6] W. G. Voss, "European union data privacy law reform: General data protection regulation, privacy shield, and the right to delisting," *Business Lawyer*, vol. 72, no. 1, pp. 221–233, 2017.

[7] M. Rafiei, L. von Waldthausen, and W. M. P. van der Aalst, "Ensuring confidentiality in process mining," in *SIMPDA*, ser. CEUR Workshop Proceedings, vol. 2270. CEUR-WS.org, 2018, pp. 3–17.

[8] H. F. Gaines, *Cryptanalysis: A study of ciphers and their solution*. Courier Corporation, 2014.

[9] L. Sweeney, "k-anonymity: A model for protecting privacy," *Int'l Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 05, pp. 557–570, 2002.

[10] C. Dwork, "Differential privacy: A survey of results," in *TAMC*. Springer, 2008, pp. 1–19.

[11] A. Monreale, D. Pedreschi, R. G. Pensa, and F. Pinelli, "Anonymity preserving sequential pattern mining," *Artificial intelligence and law*, vol. 22, no. 2, pp. 141–173, 2014.

[12] C. Cabanillas, M. Resinas, and A. Ruiz-Cortés, "Ral: A high-level user-oriented resource assignment language for business processes," in *BPM Workshops*. LNBIP, vol. 99. Springer, 2011, pp. 50–61.

[13] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed nist standard for role-based access control," *ACM TISSEC*, vol. 4, no. 3, pp. 224–274, 2001.

[14] M. zur Muehlen and R. Shapiro, "Business process analytics," in *Handbook on BPM 2*. Springer, 2010, pp. 137–157.

[15] N. Li, T. Li, and S. Venkatasubramanian, "t-closeness: Privacy beyond k-anonymity and l-diversity," in *ICDE*. IEEE, 2007, pp. 106–115.

[16] Y. Rubner, C. Tomasi, and L. J. Guibas, "The earth mover's distance as a metric for image retrieval," *International journal of computer vision*, vol. 40, no. 2, pp. 99–121, 2000.

[17] A. Augusto, R. Conforti, M. Dumas, and M. La Rosa, "Split miner: Discovering accurate and simple business process models from event logs," in *ICDM*. IEEE, 2017, pp. 1–10.

[18] A. Weijters and J. Ribeiro, "Flexible heuristics miner (FHM)," in *CIDM*. IEEE, 2011, pp. 310–317.

[19] S. J. Leemans, D. Fahland, and W. M. van der Aalst, "Discovering block-structured process models from event logs containing infrequent behaviour," in *BPM Workshops* , LNBIP, vol. 171. Springer, 2013, pp. 66–78.

[20] R. De La Briandais, "File searching using variable length keys," in *Western joint computer conference*. ACM, 1959, pp. 295–298.

[21] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, no. 8, 1966, pp. 707–710.

[22] M. De Leoni and F. Mannhardt, "Road traffic fine management process," 2015. https://data.4tu.nl/repository/uuid:270fd440-1057-4fb9-89a9-b699b47990f5

[23] J. Buijs, "Environmental permit application process ('wabo'), coselog project," 2014. https://data.4tu.nl/repository/uuid:26aba40d-8b2d-435b-b5af-6d4bfbd7a270

[24] Mannhardt, F. (Felix), "Sepsis cases - event log," 2016. https://data.4tu.nl/repository/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460

[25] A. Adriansyah, B. F. van Dongen, and W. M. van der Aalst, "Conformance checking using cost-based fitness analysis," in *EDOC*. IEEE, 2011, pp. 55–64.

[26] A. Adriansyah, J. Munoz-Gama, J. Carmona, B. F. van Dongen, and W. M. P. van der Aalst, "Measuring precision of modeled behavior," *Inf. Syst. E-Business Management*, vol. 13, no. 1, pp. 37–67, 2015.

[27] S. Spiekermann and L. F. Cranor, "Engineering privacy," *IEEE Trans. Software Eng.*, vol. 35, no. 1, pp. 67–82, 2009.

[28] J. Hoepman, "Privacy design strategies - (extended abstract)," in *IFIP TC SEC*, IFIP AICT, vol. 428. Springer, 2014, pp. 446–459.

[29] W. M. P. van der Aalst, "Responsible data science: Using event data in a "people friendly" manner," in ICEIS, LNBIP, vol. 291. Springer, 2016, pp. 3–28.

# Likelihood-based Multiple Imputation by Event Chain Methodology for Repair of Imperfect Event Logs with Missing Data

Sunghyun Sim
Department of Industrial Engineering
Pusan National University
Busan, Korea
ssh@pusan.ac.kr

Hyerim Bae
Department of Industrial Engineering
Pusan National University
Busan, Korea
hrbae@pusan.ac.kr

Yulim Choi
Department of Industrial Engineering
Pusan National University
Busan, Korea
flamethrower@pusan.ac.kr

***Abstract*— The event log recorded through an information system may be missing for various reasons, which fact may result in an imperfect event log. Performing analyses using such an imperfect event log can seriously affect the quality of the obtained results. Therefore, analyses should be performed only after processing of the missing part in the imperfect event log. In the fields of data mining and statistical analysis, various methodologies have been developed to handle data with missing values, but there are not many studies dealing with incomplete event logs that have missing data in the field of process mining. In this paper, we propose a likelihood-based Multiple Imputation by Event Chain (MIEC) method for dealing with imperfect event logs with missing data. An experiment was performed using sample event logs, and a case study was conducted using a real steel manufacturing event log to verify our method. We expect the proposed method to repair the imperfect event log to a high level and to obtain analysis result with high quality even if there are many missing data.**

***Keywords*— *Event Log quality, Repairing Event Log, Event Chain, Single Imputation by Event Relationship, Multiple Imputation by Event Chain***

## I. INTRODUCTION

In the data analysis, the quality of the data greatly affects the analysis results [1]. For this reason, research on the data preprocessing method, the outlier detection method and the missing data repairing method has been actively carried out. A methodology for dealing with missing data was first formulated by Rubin [5]. He analyzed the causes of missing values in observed data, defined the types of missing values, and further proposed a method for solving the problem using a statistical model. Since then, many researchers have tried to extend this research to techniques such as simple imputation methods [6], stochastic imputation methods [5, 8], and multiple imputation methods [9]. As a result of continuous research, various methodologies have been developed that have been combined with the Bayesian method [16, 24], the kernel method [26] and the machine-learning method [27] to achieve excellent performance in restoring missing values.

In the process mining area dealing with event log data, also, the quality of the event log is one of the key factors determining the quality of the process model. A study to deal with the missing values of the event log was first attempted by Solti [25]. Based on the General Stochastic Petri Nets and the Bayesian Network, he performed a study to repairing the value of the missing activity and the missing timestamp in the imperfect the event log. Another study related to the quality of event logs is Bose [3]. In his research, defined 4 data type (Missing Data, Incorrect Data, Imprecise Data, and Irrelevant Data) that could adversely affect process model quality. Suriadi [4] previously introduced several methods of handling imperfect event logs. They presented 11 patterns based on the 4 types of imperfect event logs and suggested ways to clean them. There are two imperfect patterns related to the dual missing event log and the following. 1) Scattered Event: when the event is lost and 2) Elusive Case: the case where the relationship between event and case is lost.

In order to handle imperfect event logs containing missing values (e.g. activity name, resource, attributes, etc), we can consider two options: multiple imputation methods developed in the statistics area and the methods of Solti [25] and Suriadi [4] developed in the process mining area. However, both methods have limitations in repairing the imperfect event log. First, we show that the multiple imputation repairing methods developed in the statistics area shows low performance since the original purpose was not for repairing event log (This problem will be shown through experiments in the case study section in Section 5). When applying the methods developed in the process mining area, the following limitations are found.

- Solti Methodology: It is the first study to reconstruct the event log with missing based on statistics. However, There is a limitation that only the missing activity and missing timestamp can be repaired even though there are various other factors such as case, resource and various attributes in the event log.
- Suriadi methodology: This is the first study to propose a handling method for 4 types of imperfect event logs. However, that's why it is difficult to say that it provides a common way to handle missing in the imperfect event log.

There are 2 challenging issues in our study. First, we improve the repairing ability about the missing activity by comparing with the previous methodology developed in the process mining area, second, develop a methodology for repairing the missing resource and the missing attributes. We have developed multiple Imputation methodologies suitable for event logs through this study. We have defined the concept of event chain to reflect the structural specificity of the event log, and also proposed the method for repairing imperfect event log containing a missing value called Single Imputation by Event Relationship and Multiple Imputation by Event Chain.

The contributions of this study are as follows: 1) to define a new concept called event chains; 2) to develop Single Imputation by Event Relationship (SIER) and Multiple Imputation by Event Chain (MIEC) algorithms; 3) to show that the proposed method outperforms various previous methodologies.

978-1-7281-0919-0/19/$31.00 ©2019 IEEE
DOI 10.1109/ICPM.2019.00013

This paper is organized as follows: Section 2 describes the background and related work. Section 3 introduces the proposed methodology for repairing imperfect event log data containing missing. In Section 4 and 5, the proposed method is verified through simulation and case study, respectively. Section 6 summarizes conclusion of our study.

## II. Background and Related Work

The data used to create statistical models in the field of instructional learning or statistics for use in data mining should satisfy the following conditions: 1) Each observation must have an independent relationship; 2) The data must consist of one or more dependent variables and independent variables. However, event logs used in process mining are slightly different from data used in existing data mining or statistical models and must satisfy the following conditions: 1) Event of Intra-Case [28] is independent, but events of Inter-Case [28] are dependent on each other; 2) There is a strong dependency between activities, resources and attribute values; 3) There is a strong dependency between the time variable and the resource used for the activity. We explain the structure of the event log in detail in the following section.

### A. Event Log

An event log is a form of data that is stored as a result of information system operation and corresponds to both structured data and semi-structured data. The event log consists of a case, an activity, a resource, a timestamp, and various attributes. The event log has been defined by researchers. Among them, we will use the definition mentioned in the study of Song [23], and the definition of the event log is as follows.

**Definition 1 (Event Logs)** Let $T$ be a set of tasks (ie., atomic workflow or process objects, also referred to as activities) and $P$ a set of originators (i.e., resource, attributes). $E=T\times P$ is the set of events, i.e., combinations of an activity and an originator. $C=E^*$ is a set of possible event sequences (traces describing a case). $L\in B(C)$ is an *event log*. Note that $B(C)$ is the set of all multi-sets over $C$. Each element of $L$ denotes a case.

### B. Type of Missing

The types of missing values were firstly classified by Rubin [5] according to the following three categories: (1) *MCAR*: Missing Completely At Random; (2) *MAR*: Missing At Random; (3) *NMAR*: Not Missing At Random [12].

*1) MCAR:* Suppose that there is a missing value in a variable $Y$ in the data. The missing data for $Y$ is called the *MCAR* if the missing probability of the data for $Y$ is not related to the value of $Y$ itself or other variables $X$ in the data. When these assumptions are met for all variables, the missing values do not affect the results of the data analysis because the missing data can be regarded as a random sample of the complete data [10, 12].

*2) MAR:* Supposed that there is data that includes at least always observed $X$ and sometimes a missing value. If we calculate the missing probability for $Y$ after controlling for $X$, and if it is not related to the $Y$ value itself, we call it random randomness. The meaning of *MAR* can be expressed as the following equation [12]:

$$Pr(Y_{missing}|Y,X) = Pr(Y_{missing}|X) \quad (1)$$

that is, the left side of Equation (1) means the missing probability of $Y$ when both $Y$ and $X$ are given, and the right side of Equation (1) means the missing probability of $Y$ when the only $X$ is given. These two probabilities are the same, which means that the loss of $Y$ is affected by $X$, while the $Y$ value itself is not.

*3) NMAR:* Other cases except for *MCAR* and *MAR*. In addition, the cause of the occurrence of the missing data occurs due to the influence of the $Y$ value itself, and the missing data of the $Y$ value does not occur at random [12].

### C. Multiple Imputation (MI)

Many previous studies have developed various methods of handling missing values including deletion [17], simple mean imputation [6, 10], conditional mean imputation [7, 18], stochastic imputation [8, 19], and multiple imputation (MI) [9, 20]. In this section, among the three imputation methods, we will describe the MI method since it is known to have good performance.

The MI method [20] is a method of estimating the missing value more accurately by evaluating each virtual complete data after creating $m$ virtual complete data by performing the single imputation method $m$ times single imputation method is performed by the following procedure. 1) Imputation step: Create a virtual complete dataset in which the missing values are replaced (the more virtual datasets are generated, the more time it takes, but the more accurate are the virtual datasets that are likely to be obtained). 2) Analysis step: Evaluate each generated virtual complete data through analysis of statistics (mean, variance, probability, etc.) for the virtual total data obtained in the imputation step. 3) Pooling step: combine the best virtual total data obtained from the evaluation with data without missing values.

There are various MI methods according to the modeling method applied when generating the virtual complete data set in the imputation step. The multiple imputation methods that are widely used include MICE [10, 11], a method using the chain method based on the fully conditional specification, KNNI (K-Nearest-Neighbor Imputation) [23], RFI (Random Forest-based Multiple Imputation) [22], EMBI (Expectation-Maximization-based Bootstrapping Imputation) [21].

### D. Handling of Imperfect Event Log

A study on the handling of missing data in the area of event logs was first proposed by Solti [25]. He derived Bayesian Network models and Stochastic Petri Net models from incomplete event logs and proposed a Generally Distributed Transition durations (GDT) model. Further, through the proposed method, he repaired the missing activity and the missing timestamp from the imperfect event log.

## III. Proposed Method

In the imperfect event log, not only the activity information is missing, but resource information and other attributes may also be missing. For this reason, this study proposes a repairing method that can enhance the repairing ability of the missing activities and repair the missing resources and the missing attributes. In addition, we can have

both numeric values and categorical values in the case of an attribute. We focus on repairing the character-type missing value. The proposed repairing method for imperfect event logs consists of the 4 following steps: 1) Log Analysis; 2) Single Imputation by Event Relationship; 3) Multiple Imputation by Event Chain; 4) Verification. Fig. 1. shows the overall framework of the proposed method

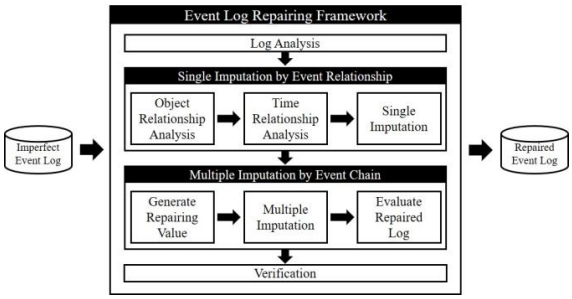



Fig. 1. Overall Framework for Imperfect Event Log Repairing

In Fig. 2, in order to explain our method effectively, we illustrate an example event log, which is introduced in the IEEE TF at the Process Mining site [29].

| | Case | Activity | Resource | Part Desc. | Start Time | End Time |
|---|---|---|---|---|---|---|
| $e_1$ | $Case_1$ | Turning & Milling | Machine 4 | Cable Head | 2012-01-29 23:24 | 2012-01-30 05:43 |
| $e_2$ | $Case_1$ | Turning & Milling | Machine 4 | Cable Head | 2012-01-30 05:44 | 2012-01-30 06:42 |
| $e_3$ | $Case_1$ | Turning & Milling | Machine 4 | Cable Head | 2012-01-30 06:59 | 2012-01-30 07:21 |
| $e_4$ | $Case_1$ | Turning & Milling | Machine 4 | Cable Head | 2012-01-30 07:21 | 2012-01-30 10:58 |
| $e_5$ | $Case_1$ | Turning & Milling Q.C | Quality Check 1 | Cable Head | 2012-01-31 13:20 | 2012-01-31 14:50 |
| $e_6$ | $Case_1$ | Laser Marking | Machine 7 | Cable Head | 2012-02-01 08:18 | 2012-02-01 08:27 |
| $e_7$ | $Case_1$ | Lapping | Machine 1 | Cable Head | 2012-02-14 00:00 | 2012-02-14 01:15 |
| $e_8$ | $Case_1$ | Lapping | Machine 1 | Cable Head | 2012-02-14 00:00 | 2012-02-14 01:15 |
| $e_9$ | $Case_1$ | Lapping | Machine 1 | Cable Head | 2012-02-14 09:05 | 2012-02-14 10:20 |
| $e_{10}$ | $Case_1$ | Lapping | Machine 1 | Cable Head | 2012-02-14 09:05 | 2012-02-14 09:38 |
| $e_{11}$ | $Case_1$ | Round Grinding | Machine 3 | Cable Head | 2012-02-14 09:13 | 2012-02-14 13:37 |
| $e_{12}$ | $Case_1$ | Round Grinding | Machine 3 | Cable Head | 2012-02-14 13:37 | 2012-02-14 15:27 |
| $e_{13}$ | $Case_1$ | Final Inspection Q.C. | Quality Check 1 | Cable Head | 2012-02-16 06:59 | 2012-02-16 07:59 |
| $e_{14}$ | $Case_1$ | Final Inspection Q.C. | Quality Check 1 | Cable Head | 2012-02-16 12:11 | 2012-02-16 16:12 |
| $e_{15}$ | $Case_1$ | Final Inspection Q.C. | Quality Check 1 | Cable Head | 2012-02-16 12:43 | 2012-02-16 13:58 |
| $e_{16}$ | $Case_1$ | Packing | Packing | Cable Head | 2012-02-17 00:00 | 2012-02-17 01:00 |
| $e_{17}$ | $Case_2$ | Turning & Milling | Machine 9 | Spur Gear | 2012-01-17 07:01 | 2012-01-17 11:05 |
| $e_{18}$ | $Case_2$ | Turning Q.C. | Quality Check 1 | Spur Gear | 2012-01-17 11:06 | 2012-01-17 11:15 |
| $e_{19}$ | $Case_2$ | Turning & Milling | Machine 9 | Spur Gear | 2012-01-17 19:24 | 2012-01-17 20:01 |
| $e_{20}$ | $Case_2$ | Turning & Milling | Machine 9 | Spur Gear | 2012-01-17 20:01 | 2012-01-17 23:43 |
| ... | ... | ... | ... | ... | ... | ... |

Fig. 2. Production event log provided by IEEE TF at Process Mining site

## A. Log Analysis

In the log analysis step, the types of events with missing values are summarized. In Fig. 3, for example, in the case of event $e_1$, it is a type with only an activity missing, and event $e_2$ is a type with only a resource missing. However, event $e_5$ is a type that has missing activity, resource, and attribute values.

| | Case | Activity | Resource | Part Desc. | Start Time | End Time |
|---|---|---|---|---|---|---|
| $e_1$ | $Case_1$ | ① | Machine 4 | Cable Head | 2012-01-29 23:24 | 2012-01-30 05:43 |
| $e_2$ | $Case_1$ | Turning & Milling | ② | Cable Head | 2012-01-30 05:44 | 2012-01-30 06:42 |
| $e_3$ | $Case_1$ | Turning & Milling | Machine 4 | ③ | 2012-01-30 06:59 | 2012-01-30 07:21 |
| $e_4$ | $Case_1$ | Turning & Milling | Machine 4 | Cable Head | 2012-01-30 07:21 | 2012-01-30 10:58 |
| $e_5$ | $Case_1$ | ④ | ⑤ | Cable Head | 2012-01-31 13:20 | 2012-01-31 14:50 |
| ... | ... | ... | ... | ... | ... | ... |

Fig. 3 Simple sample log with missing data in the production event log.

To apply our imputation technique, we need to replace the missing values with proper values in the imperfect event log. In this situation, it can be intuitively understood that when only one missing value occurs in one event, it can be more accurately repaired than when multiple missing values occur in one event. In addition, imputation using various relationships between elements (between objects or between time and sequence) in an event can increase repairing accuracy. The role of the log analysis step is used to determine whether to use single imputation (SI) or multiple imputation (MI) to impute a value for each event when repairing the event log. In the next section, we will explain the procedure of single imputation and when to apply this method.

## B. Single Imputation by Event Relationship

We have developed a new method named Single Imputation by Event Relationship (SIER) that applies single imputation using various relationships between an element in events. The SIER algorithm needs to conduct two types of relationship analysis: Object Relationship Analysis (ORA) and Time Relationship Analysis (TRA).

Since the event log is structured or semi-structured, there are relationships among entities. In other words, when we impute the missing value, we can use cardinality relationships among them as summarized in Table 1.

Table 1 Notation for Event Object Relationship

| Object Relationship | Correspond | | | |
|---|---|---|---|---|
| | **1:1** | **1:*n*** | ***n*:1** | ***n*:*m*** |
| Ac-Re (Activity-Resource) | $Ac_1Re_1$ | $Ac_1Re_n$ | $Ac_nRe_1$ | $Ac_nRe_m$ |
| Re-At (Resource-Attribute) | $Re_1At_1$ | $Rc_1At_n$ | $Re_nAt_1$ | $Re_nAt_m$ |
| Ac-At (Activity-Attribute) | $Ac_1At_1$ | $Ac_1At_n$ | $Ac_nAt_1$ | $Ac_nAt_m$ |

The cardinality can be obtained by analyzing the event occurrence matrix shown in Fig. 4. From the artificial data in the figure, we can extract the 3 relation types between activity and resource entities.

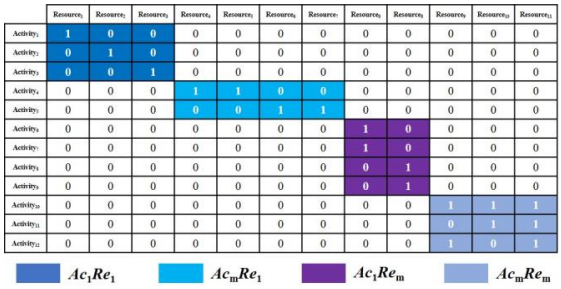



Fig. 4. Artificial sample object (Activity-Resource) relation matrix.

Next, SIER carries out the time relationship analysis. There are two types of time relationship: time-dependent and time-independent. In time-dependent relationships, in a single time period, there can be multiple occurred events. In time-independent relationships, only one event can occur in a single time period. Fig. 5. shows the relationship between these two types. Figs. 5.a and 5.b show event logs that

illustrate time-dependent and time-independent relationships, respectively. Fig 5.a shows that a multi-event occurs in a single time period when performing an activity called "Lapping." In this case, we can see that the event occurs regardless of time. On the other hand, Fig. 5.b shows that there is only one event in the single time period when performing "Fix EDM." In other words, in this case, it can be said that an event has a time-dependent relationship.

| | Case | Activity | Resource | Part Desc. | | Start Time | End Time |
|---|---|---|---|---|---|---|---|
| $e_{147}$ | $Case_{108}$ | Lapping | Machine 1 | Bushing | | 2012-03-01 02:35 | 2012-03-01 03:50 |
| $e_{148}$ | $Case_{108}$ | Lapping | Machine 1 | Bushing | | 2012-03-01 02:36 | 2012-03-01 03:28 |
| | … | … | … | … | … | … | … |
| $e_{236}$ | $Case_{110}$ | Lapping | Machine 1 | Ballnut | | 2012-03-22 12:15 | 2012-03-22 14:33 |
| $e_{237}$ | $Case_{110}$ | Lapping | Machine 1 | Ballnut | | 2012-03-22 13:42 | 2012-03-22 16:22 |
| … | … | … | … | … | | … | … |

Fig. 5.a: Time-independent relationship case

| | Case | Activity | Resource | Part Desc. | | Start Time | End Time |
|---|---|---|---|---|---|---|---|
| $e_{887}$ | $Case_{148}$ | Fix EDM | Machine 17(Sinking) | Piston | | 2012-01-17 08:00 | 2012-01-17 15:00 |
| | … | … | … | … | | … | … |
| $e_{3934}$ | $Case_{70}$ | Fix EDM | Machine 17(Sinking) | Spindle | … | 2012-03-04 23:59 | 2012-03-05 02:00 |
| $e_{3944}$ | $Case_{70}$ | Fix EDM | Machine 17(Sinking) | Spindle | | 2012-03-07 04:00 | 2012-03-07 05:00 |
| $e_{3966}$ | $Case_{70}$ | Fix EDM | Machine 17(Sinking) | Spindle | | 2012-03-26 09:30 | 2012-03-26 10:30 |
| … | … | … | … | … | | … | … |

Fig. 5.b: Time-dependent relationship case

Fig. 5. Examples of time-dependent and time-independent relationships

After we analyze the relationships among entities, we can decide whether to use the SI method or the MI method. Using the ORA result, if the relationship has 1:1 cardinality ($Ac_1Re_1$, $Re_1At_1$, $Ac_1At_1$), we can impute any of the entities. Otherwise, if the relationship has 1:n and n:1 cardinality, we can impute 1-side missing value using single imputation. Finally, if the relationship has the n:m relation ($Ac_nRe_m$, $Re_nAt_m$, $Ac_nAt_m$), we have to carry out the MI method. Using the TRA result, if it is a time-dependent relationship, we have to conduct single imputation for the missing value. Otherwise, we need to conduct the MI method.

In the example in Fig. 3, since "Machine 4" and "Turning & Milling" are related to the $Ac_1Re_1$ relationship, they can be repaired with "Turning & Milling" and "Machine 4," respectively to ① and ② through single imputation. On the other hand, the values of all part Desc. in this event log should use multiple imputations because ③ has the *n*:*m* relationship for all activity values and all resource values. In addition, since ④ and ⑤ missing values in $e_5$ contain multiple missing values in one event, the missing values are restored using an MI method. We will explain the MI method for repairing an event log in the next section.

### C. Multiple Imputation by Event Chain

The MI steps are used to repair 1:*n*, *n*:1 (n-side value), and *n*:*m* relationships after applying the SIER algorithm. There are 3 steps as follows: 1) Generate repair value; 2) Multiple imputation (MI); 3) Estimate repaired log. In order to successfully apply the MI technique, which is the core technology of this stage, we decided to define a new random variable including sequential information between events. For this purpose, we newly defined the concept of the event chain.

Generally, in set theory, a chain is defined as a sequence that is separated from a set in the same length by a subset having the same length. Each event in the event log can also be represented by an event set in which the sequence exists, and can be further separated into subsets having the same length. We define a subset that satisfies these characteristics as Event Chain (see below), and Fig. 6. shows the process of creating an event chain from a simple sample event log.

**Definition 2 (Event Chain)** In each case, there is a start event label, and there is an end event label. Define event chain when combining prior events ($ec_{j-}$), a current event ($ec_j$), and a posterior event ($ec_{j+}$) in one event chain ($EC=\{ec_j | ec_{j-}, ec_j, ec_{j+}\}$) in the same case (the $j^{th}$ event in the same case). *EC* consists of activity chain, resource chain, and attribute chain.

| | Case | Activity | Resource | Part Desc. | | Start Time | End Time |
|---|---|---|---|---|---|---|---|
| … | … | … | … | … | | … | … |
| $e_2$ | $Case_1$ | Turning & Milling | Machine 4 | Cable Head | | 2012-01-30 05:44 | 2012-01-30 06:42 |
| $e_3$ | $Case_1$ | Turning & Milling Q.C | Quality Check 1 | Cable Head | … | 2012-01-30 06:59 | 2012-01-30 07:21 |
| $e_4$ | $Case_1$ | Laser Marking | Machine 7 | Cable Head | | 2012-01-30 07:21 | 2012-01-30 10:58 |
| $e_5$ | $Case_1$ | Lapping | Machine 1 | Cable Head | | 2012-01-31 13:20 | 2012-01-31 14:50 |
| … | … | … | … | … | | … | … |

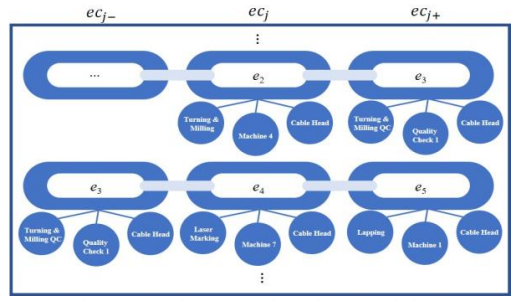



Fig. 6. Event chain generation using example log

Using the newly proposed event chain, we, in turn, propose a MI method called Multiple Imputation by Event Chain (MIEC). The MIEC algorithm architecture and flowchart are shown in Figs. 7 and 8, respectively.

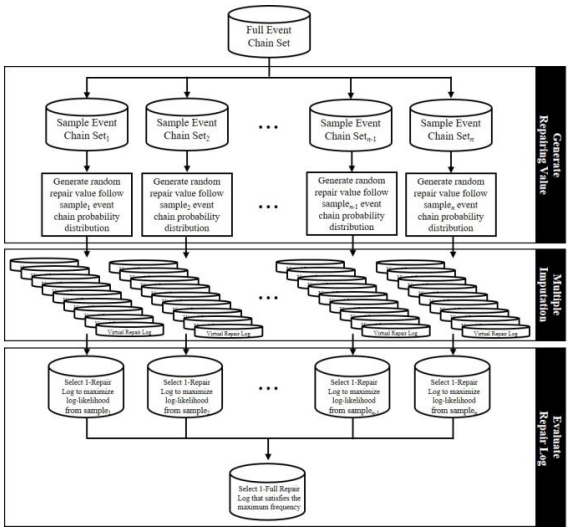



Fig. 7. The architecture of the MIEC algorithm

To perform the MI method, we first have to generate a repair value. In order to generate the repair value, first, a set of random sample event chains $EC_S$ having S sizes is extracted from ***EC*** from which the missing part is removed.

Then, the event chain occurrence probability distribution $f$ of the sample is estimated from the extracted samples $EC_S$, and a repairing value $R$ is randomly generated by the number of the missing value so as to follow the probability distribution $f$.

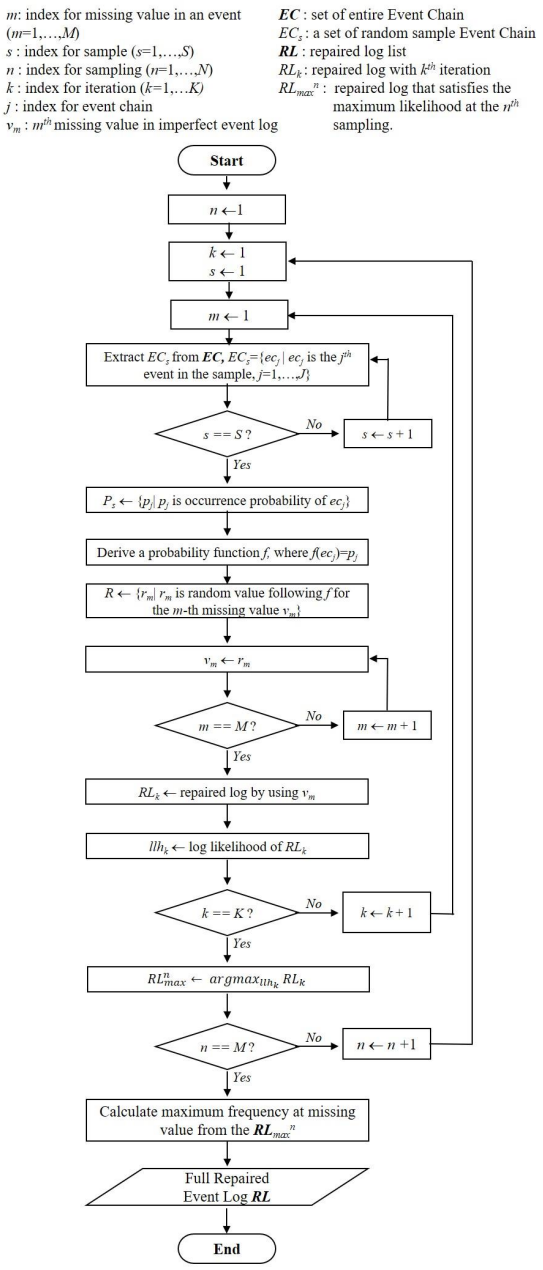



Fig. 8 Flowchart of MIEC algorithm

After generating the repair value $R$, the MI method is performed based on it. In this case, imputation is performed while changing the order of $R$ values at the missing position by repeating the predetermined number of iterations, and log-likelihood is calculated for the repaired log at the end of each iteration. At the end of each iteration, we obtain the repaired event log, and when the specified repetition number ends, we obtain the repaired event log for the number of iterations. At this point, we will select one repair log that can maximize the log-likelihood, which is the event log repaired to follow well the random sample event chain $EC_s$. We perform a sufficient number of samplings in the same manner as above and collect repaired event logs $RL$ that closely follow the occurrence probability distribution $f$ of each sample. In the last stage of the MIEC, the $RL_{max}^n$ obtained through a sufficient number of sampling steps is analyzed. In this step, only one repair value is selected for each position in consideration of which value is repaired most (maximum frequency) at the position where each missing value occurs, and a full repaired log is obtained.

### D. *Verification*

In the verification step, the object relationship analysis and the time relationship analysis are performed once more using the final repair log, and the step is compared with the initial obtained value. If the relationship between event entities in the event log before and after repair does not match, the repaired event log is rejected and the MIEC algorithm is repeated. If the event relationship matches, all repair procedures are terminated by accepting the repaired event log.

## IV. Experiment

### A. *Experiment Procedure*

In Section III, we proposed a method for repairing imperfect event logs using multiple imputations by the event chain algorithm based on likelihood. In this section, the performance of the proposed method is verified by following experiments. The experiment used the production process event log data included in the IEEE TF at the Process Mining site (event log data including 4544 events, 225 cases, 55 activities, 31 machine resources, 43 attributes (part Desc)). Furthermore, to analyze the accuracy and the time performance according to the data size, the experiment is performed by increasing the data size from 2 times (9088 events, 450 cases) to 5 times (22720 events, 1125 cases) using production process event log data.

For our experiments, we generated missing values randomly in the columns of Activity, Resource, and Attribute at the rate of 5% to 30% (at 5% intervals). For the imperfect event log, we applied our method to repair the log. We conducted repair for 30 iterations and calculated the accuracy by comparing the repaired log with the original log.

### B. *Experiment Result without Increasing Data Size*

The results of our experiment are summarized in Table 2 and Fig. 7 by bar chart and confusion matrix using the event log without increasing the data size, respectively.

Table 2 Experimental Results without increasing the data size

| Missing Rate | Total | Activity | | Resource | | Attribute | |
|---|---|---|---|---|---|---|---|
| | $\mu$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ |
| 5% | **98.5%** | 99.1% | 0.41% | 97.6% | 0.97% | 98.7% | 0.59% |
| 10% | **96.5%** | 97.3% | 1.33% | 95.3% | 1.38% | 97.0% | 0.73% |
| 15% | **93.4%** | 92.5% | 1.45% | 92.6% | 1.57% | 95.2% | 1.28% |
| 20% | **87.4%** | 87.6% | 1.59% | 84.5% | 1.95% | 90.1% | 1.66% |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 25% | **83.2%** | 83.7% | 1.71% | 79.8% | 2.12% | 86.2% | 2.13% |
| 30% | **76.2%** | 78.8% | 2.02% | 70.2% | 2.53% | 79.6% | 2.34% |

The results showed 93% repairing accuracy in the case of 15% of random missing values in the columns of activity, machine resource, and attribute (part Desc). In the case where we had more missing values (at the rate of 25%), we could achieve 80% accuracy.

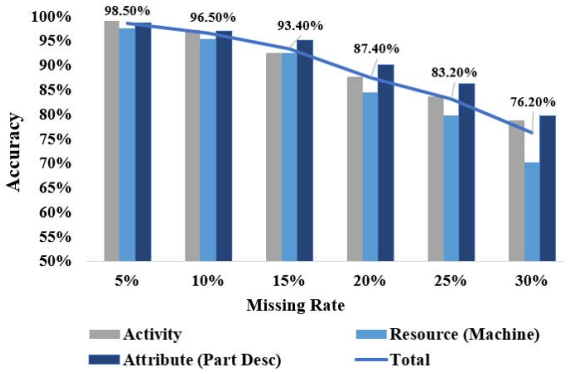



Fig. 9. Bar chart and line chart in experiments

We confirmed that the repairing ability of the resource was lower than those of the activity and the attribute. The reason for this is that resources are often on the *n*-side rather than the other two entities, and more often they have a time-independent relationship. In this case, the missing resources must be repaired using multiple imputations only. This part is expected to improve the performance of the multiple imputations parts in future studies to obtain better repairing ability naturally.

### C. *Experiment Result with Increasing Data Size*

Table 4 shows the result of repairing the missing value while increasing the size of the event logs size form 2 times to 5 times. The experimental results show that if the ratio of missing value is the same, even if the size of the event log increases, the accuracy does not decrease significantly. Furthermore, Table 4 shows that when the missing ratio is less than or equal to 20%, while the event log size is increased, the repairing accuracy difference is happened only 2% to 4%.

Table 4 Repairing Accuracy with increasing the data size

| Missing Rate | Data size | | | |
|---|---|---|---|---|
| | 9,088 Events 450 Cases | 13,632 Events 675 Cases | 18,176 Events 900 Cases | 22,720 Events 1,125 Cases |
| 5% | 97.9% | 97.3% | 96.8% | 96.1% |
| 10% | 96.0% | 95.3% | 94.8% | 94.1% |
| 15% | 92.3% | 91.6% | 90.1% | 89.7% |
| 20% | 86.9% | 86.1% | 85.3% | 84.9% |
| 25% | 81.2% | 79.3% | 77.2% | 75.8% |
| 30% | 74.7% | 72.7% | 70.9% | 68.5% |

Next, the time performance is performed according to the event log size, and Table 10 shows the result. Each time you increase the size of the data, the time performance increases at a certain rate. In particular, the 5 times (22,720 events, 1,125 cases) in the size of the event log has increased by 3% to 4% overall. This eventually increases the time performance due to the number of times of possible imputations combination increases at the MIEC step.

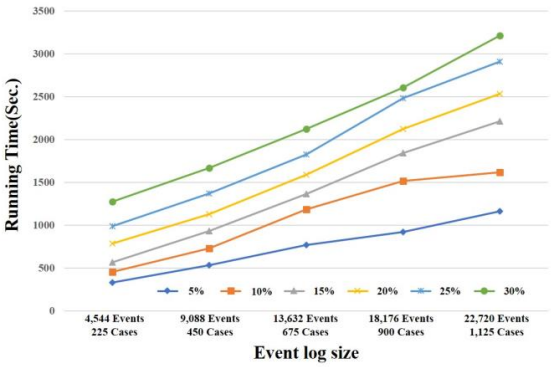



Fig. 10 Result of Time Performance Analysis

## V. CASE STUDY FOR COMPARING THE PREVIOUS METHOD

In this section, a case study was conducted using real process data from a Korean steel company to check the applicability of actual problems.

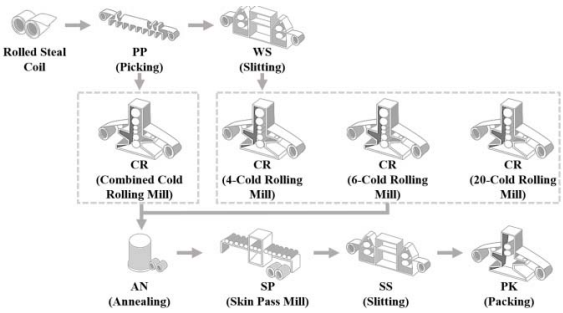



Fig. 11. Manufacture process of the steel company in Korea

The main process of this company is as follows. 1) The coil-shaped iron plate is oxidized (PP). 2) It is cut to a size suitable for the order (WS). 3) The shape and thickness of the surface are adjusted by cold rolling to suit the product type and application (CR). 4) The properties of the iron are improved through annealing (AN). 5) The processes of surface treatment (SP) and of finely cutting the end of the steel plate (SS) are conducted. 6) The process of packaging products (PK) is conducted. Fig. 8. shows the process of manufacture described above. From the event logs containing 21201 events, 2391 cases, 12 activities, 37 machines (resource), and 556 order numbers (attribute). We performed repairs using the proposed method after randomly generating missing values from 5% to 20% at 5% intervals. Furthermore, in this case study, two comparisons were made to compare with various existing methodologies.

- Comparison Experiments 1 (CE1): The event log repair method based on the GDT model is a method designed to recover the missing activity and the missing timestamp. For this reason, a comparison of the repairing ability of the GDT model with the proposed method was conducted for the missing activity value only.
- Comparison Experiments 2 (CE2): A comparison of the repairing ability of the previous method and the proposed method on imperfect event logs with missing activity, missing resource, and missing attributes

Tables 5 and 6 show the results of CE1 and CE2, respectively. First of all, we can see from Table 5 that our proposed method shows about 7% to 10% of better repairing ability than the GDT model, which was developed to restore missing activity in the process mining field. In addition, Table 6 shows that the method proposed in this study outperforms the previous methods by other research in CE2, which compares the five missing data-repairing methodologies developed in the data mining and process mining fields with the newly proposed methodology.

Table 5. Results of CE1

| Missing Rate | GDT | MIEC |
|---|---|---|
| 5% | 88.3% | **96.2%** |
| 10% | 84.4% | **95.0%** |
| 15% | 82.6% | **91.5%** |
| 20% | 77.6% | **84.4%** |

Table 6. Results of CE2

| Missing Rate | MICE | EMBI | RFI | KNNI | GDT | MIEC |
|---|---|---|---|---|---|---|
| 5% | 48.9% | 51.3% | 60.1% | 44.4% | 72.3% | **93.2%** |
| 10% | 44.3% | 45.5% | 59.8% | 42.1% | 65.4% | **91.0%** |
| 15% | 34.7% | 36.9% | 50.3% | 40.3% | 60.8% | **88.7%** |
| 20% | 28.5% | 30.1% | 47.7% | 32.2% | 53.3% | **80.4%** |

## VI. Conclusion

In this paper, we proposed a new method to repair imperfect event logs with missing activities, missing resources, and missing attributes. In order to repair the missing values of imperfect event logs, our proposed method was conducted by following 4 steps. 1) Imperfect event logs analysis step; 2) SIER (Simple Imputation by Event Relationship) step; 3) MIEC (Multiple Imputation by Event Chain) step; 4) Verification step. Notably, this study is the first to apply the Multiple Imputation (MI) methodology event log, which is mainly handled in the fields of statistics and data mining.

We have developed new algorithms called SIER and MIEC to repair the missing values of imperfect event logs. The SIER algorithm uses Single Imputation which processes the missing values through the event relationships (object relationship and time relationship). For applying MI method in the event log, we introduce a new definition called; the event chain to develop a MI method suitable for event logs. Further, we developed a method called MIEC (Multiple Imputation by Event Chain) using the concept of the event chain and devised a methodology for restoring missing activities, missing resources and missing attributes composed of character-type values.

Our methodology was verified as having a good repairing ability through the experiments. Moreover, a case study using an actual steel manufacturing event log showed that it is superior to previous methodologies.

The contributions of our study are as follows. First, the concept of the event chain is newly defined, and the MI techniques that can be applied to the event log are developed. Second, the resilience of missing activity values of imperfect event logs is improved to a higher level relative to those of the existing methods. Finally, a methodology, which was can repair the missing values of resources and various attribute values that were not previously considered, was developed.

We expect that even if there is an imperfect event log with missing values, our method is able to provide a restoration level similar to the original event log. Furthermore, by improving the quality of the event log, we can use most of process mining techniques without great loss of accuracy although there is a serious defect in the log.

## Acknowledgment

This work was partially supported by the MSIT (Ministry of Science and ICT), Korea, under the Grand Information Technology Research Center support program (IITP-2018-2016-0-00318) supervised by the IITP (Institute for Information & communications Technology Promotion)

## References

[1] L. Cai, and Y. Zhu, “The challenges of data quality and data quality assessment in the big data era,” Data Science Journal, vol. 14, 2015.

[2] W. M. P. Van der Aalst, “Process Discovery: An Introduction. In Process Mining”, Springer, Berlin, pp. 125-156. 2011.

[3] J. C. Bose, R. S. Mans, W.M.P Van der Aalst, “Wanna improve process mining results? It's high time we consider data quality issues seriously,” in Proceedings of the IEEE CIDM, IEEE, Singapore, pp. 127-134, 2013.

[4] S. Suriadi, R. Andrews, A. H. ter Hofstede, M. T. Wynn, “Event log imperfect patterns for process mining: Towards a systematic approach to cleaning event logs,” Information Systems, vol. 64, pp 132-150, 2017.

[5] D. B. Rubin, “A non-iterative algorithm for least squares estimation of missing values in any analysis of variance design,” Applied Statistics, pp 136-141, 1972.

[6] G. Fortino, P. Trunfio, “Internet of things based on smart objects: Technology,” middleware and applications. Springer, Science & Business Media, 2014.

[7] , A. R. T. Donders, G. J. Van Der Heijden, T. Stijnen, K. G. Moons, “A gentle introduction to imputation of missing values,” Journal of clinical epidemiology, vol. 59(10), pp 1087-1091, 2006.

[8] B. Kitts, D. Freed, M. Vrieze, “Cross-sell: a fast promotion-tunable customer-item recommendation method based on conditionally independent probabilities,” In Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 437-446. 2000.

[9] F. M. Shrive, H. Stuart, H. Quan, W. A. Ghali, “Dealing with missing data in a multi-question depression scale: a comparison of imputation methods,” BMC medical research methodology, vol. 6(1), 57, 2006.

[10] D. B. Rubin, “Multiple imputation for nonresponse in surveys,”. John Wiley Sons, vol. 81, 2004.

[11] S. Van. Buuren, K. Groothuis-Oudshoorn, "MICE: multivariate imputation by chained equations in R," Journal of statistical software, pp 1-68, 2010.

[12] J. L. Schafer, J. W. Graham, "Missing data: our view of the state of the art," Psychological methods, vol. 7(2), 147, 2002.

[13] P. Schmitt, J. Mandel, M. Guedj, "A comparison of six methods for missing data imputation," J Biomet Biostat. vol. 6(1), pp 1-6, 2015.

[14] W. M. P. Van der Aalst, H. A. Reijers, A. J. Weijters, B. F. Van Dongen, A. A. De Medeiros, M. Song, H. M. W. Verbeek, "Business process mining: An industrial application," Information Systems, vol. 32(5), pp 713-732, 2007.

[15] W. M. P. Van der Aalst, "Process mining: data science in action," Springer, 2016.

[16] R. J. Little, D. B. Rubin, "Statistical analysis with missing data," John Wiley Sons, vol. 333, 2014.

[17] D. B Rubin, "Inference and Missing Data," Biometrika, vol. 63(3), pp 581-592, 1976.

[18] G. L. Schlomer, S. Bauman, N. A. Card, "Best practices for missing data management in counseling psychology," Journal of Counseling psychology, vol. 57(1), 1, 2010.

[19] P. D. Allison, "Missing data techniques for structural equation modeling," Journal of abnormal psychology, vol. 112(4), 545, 2003.

[20] P. D. Allison, "Missing data: Quantitative applications in the social sciences," British Journal of Mathematical and Statistical Psychology, vol. 55(1), pp 193-196, 2002.

[21] L. R. Landerman, K.C. Land, C. F. Pieper, "An empirical evaluation of the predictive mean matching method for imputing missing values," Sociological Methods Research, vol. 26(1), pp 3-33, 1997.

[22] N. J. Horton, K. P. Kleinman, "Much ado about nothing: A comparison of missing data methods and software to fit incomplete data regression models," The American Statistician, vol. 61(1), pp 79-90, 2007.

[23] M. Song and W. M. Van der Aalst, "Towards comprehensive support for organizational mining," Decision Support Systems, vol. 46(1), pp 300-317, 2008.

[24] S. Oba, M. A. Sato, I. Takemasa, M. Monden, K. I. Matsubara and S. Ishii, "A Bayesian missing value estimation method for gene expression profile data," Bioinformatics, vol. 19(16), pp 2088-2096 2003.

[25] A. Rogge-Solti, R. S. Mans, W. M. Van der Aalst and M. Weske, "Repairing event logs using stochastic process models," Universitätsverlag Potsdam., vol. 78, 2013

[26] Zhang, Dao-Qiang, and C. Song-Can "Clustering incomplete data using kernel-based fuzzy c-means algorithm," Neural processing letters, vol18(3), pp 155-162, 2003

[27] J. M. Jerez, I. Molina, P. J. García-Laencina, E. Alba, N. Ribelles, M. Martín and L. Franco, "Missing data imputation using statistical and machine learning methods in a real breast cancer problem," Artificial intelligence in medicine, vol. 50(2), pp 105-115, 2010

[28] A. Senderovich, C. Di Francescomarino, C. Ghidini, K. Jorbina and F. M. Maggi, "Intra and inter-case features in predictive process monitoring: A tale of two dimensions," In International Conference on Business Process Management, Springer, Cham, pp. 306-323, 2017

[29] IEEE Task Force on Process Mining - Event Logs, "http://data.4tu.nl/repository/uuid:68726926-5ac5-4fab-b873ee76ea41 2399"

# Scalable Mixed-Paradigm Trace Clustering using Super-Instances

Pieter De Koninck and Jochen De Weerdt

***Abstract*— In process mining, one is often confronted with datasets that contain high degrees of control-flow variation. This causes the results of subsequent process mining steps to be less accurate or harder to understand. Trace clustering, or dividing the log into more homogeneous groups, can help limit this issue. Trace clustering techniques are commonly subdivided into two types: distance-based techniques and model-driven approaches. In this paper, a novel trace clustering technique is presented, that combines aspects of both paradigms. The core idea is to first learn so-called super-instances using a simple distance-driven technique and subsequently apply a model-driven technique to cluster the super-instances. Our technique not only shows qualitative improvements of the obtained clusterings, but also significantly improves the scalability, as shown in an experimental evaluation using real-life event logs.**

## I. INTRODUCTION

Despite the demonstrated usefulness of process discovery in all kinds of business environments, it has been shown that application to real-life data is often made difficult by the large variety in observed behavior caused by the multitude of sequential orders in which activities were executed. In some cases, this variety might stem from a so-called long tail of exceptional behavior, however, event logs originating from flexible environments naturally contain higher amounts of process variants. Partitioning algorithms operating at the trace level have been proposed under the umbrella term of *trace clustering*. These techniques are an interesting approach to deal with the problem that default process discovery techniques usually churn out inaccurate or totally uninterpretable results for such event logs.

In this paper, we go beyond the state of the art in trace clustering by proposing TraCluSI (Trace Clustering using Super-Instances), a new trace clustering technique that combines the two dominant trace clustering paradigms, i.e. model-based clustering and distance-driven clustering. TraCluSI makes use of so-called super-instances, which are obtained by overclustering the dataset using a distance-driven approach. In a next step, these super-instances are provided to a model-driven clustering technique. As such, our approach is not only capable of improving the qualitative results of clusterings, but also addresses the scalability problem, which many trace clustering techniques suffer from.

Our paper is structured as follows. First, Section II discusses related work. In Section III, our technique is introduced and subsequently evaluated in Section IV. Finally, Section V concludes the paper.

P. De Koninck and J. De Weerdt are with Faculty of Economics and Business, Research Centre for Information Systems Engineering (LIRIS), KU Leuven, 3000 Leuven, Belgium. Email: pieter.dekoninck@kuleuven.be and jochen.deweerdt@kuleuven.be

## II. RELATED WORK

Numerous trace clustering techniques exist within the process mining field. For a recent comparative assessment, see e.g. [1]. Table I presents a basic overview of trace clustering techniques and their corresponding paradigm. Trace clustering techniques often translate the problem to a propositional setting by featurizing the process instances, and subsequently applying a traditional data clustering technique. Given that they usually employ distance-based data clustering algorithms, we refer to them as "distance-based". An entirely different set of trace clustering techniques are model-driven, i.e. they rely either on some temporal probabilistic model like a Hidden Markov Model, or on actual process models (in the form of Petri nets, dependency graphs, or other representations).

TABLE I
AN OVERVIEW OF TRACE CLUSTERING TECHNIQUES AND THEIR CORRESPONDING PARADIGM

| Author | Distance-based | Model-driven |
|---|---|---|
| Greco et al. [2] | ✔ | |
| Song et al. [3] | ✔ | |
| Bose et al. [4] | ✔ | |
| Bose et al. [5] | ✔ | |
| Delias et al. [6] | ✔ | |
| Appice and Malerba [7] | ✔ | |
| Evermann et al. [8] | ✔ | |
| Ferreira et al. [9] | | ✔ |
| Folino et al. [10] | | ✔ |
| De Weerdt et al. [11] | | ✔ |
| Chatain et al. [12] | | ✔ |
| De Koninck et al. [13] | | ✔ |
| *TraCluSI* | ✔ | ✔ |

*1) Distance-based clustering:* The simplest idea for clustering traces relies on the use of traditional data clustering techniques [14]. Hereto, one should convert an event log, i.e. a set of traces, into an attribute-value dataset. In a pioneering study, Greco et al. [2] proposed to featurize traces using frequent k-grams. In [3], this idea was extended by the proposal of "profiles" to determine the vector associated with each process instance. Similarly, Bose et al. [5] propose a refinement of the technique presented in [2] by making use of conserved patterns (e.g. Maximal, Super Maximal, and Near Super Maximal Repeats). Their implementation applies hierarchical clustering instead of k-means.

In [4], Bose and van der Aalst diverge from the use of an attribute-value feature space, but instead regard log traces as strings over the alphabet of task labels and as such

978-1-7281-0919-0/19/$31.00 ©2019 IEEE
DOI 10.1109/ICPM.2019.00014

calculate the distance between traces directly by proposing a tailored string edit distance. A similar alignment-based technique is proposed in [8]. Furthermore, in [6] a more robust spectral learning approach was proposed, in which the similarity metric is adjusted in the face of significant deviating behaviour. Finally, in [7], the idea of a co-training strategy was deployed based on multiple views: separate clusterings are created using instance similarity based on the activities, resources, sequences and timing aspects of the process instances, and then combined to create a single multiple-view clustering.

*2) Model-driven trace clustering:* The second paradigm can be best described as model-driven. Inspired by [15] in the area of web usage mining, Ferreira et al. [9] propose to cluster sequences by learning a mixture of first-order Markov models using the Expectation-Maximization (EM) algorithm. Also in [10], Markov chain-based clustering is used. Furthermore, De Weerdt et al. [11] propose the *ActiTraC*-algorithm, which is an active learning-inspired trace clustering technique that strives to optimise the combined fitness of the underlying process models. Also in [12], a process model-driven technique is proposed, which relies on the concept of alignments to construct models that serve as centroids. Finally, expert-driven trace clustering has received attention recently as well. In [13], a trace clustering algorithm inspired by semi-supervised clustering is developed that takes both expert information and process model quality into account while grouping the traces.

*3) Clustering with representatives:* The concept of using representatives to speed up clustering is not new. Generally speaking, most agglomerative hierarchical clustering techniques contain the notion of representing larger sets of instances with smaller subsets, for example when calculating inter-cluster distances. More specifically, CURE [16], or clustering using representatives, extends this notion by proposing a specific approach for selecting representative instances and for merging clusters. More closely related to the approach presented in this paper is [17]. There, the authors propose using representatives, which they call "super-instances" to improve active constraint selection in a constrained data clustering setting.

## III. TraCluSI: Trace Clustering using Super-Instances

### A. Preliminaries

An event log is a collection of traces (also called process instances). Each trace contains an identifier, a sequence of events, and optional other attributes. Events can be ordered simply as a sequence, but are typically denoted based on a timestamp.

*Definition 3.1 (Event):* An event is a tuple e=(p,$\tau$,a,$x_1$,...,$x_n$) where p is the identifier of the process instance it belongs to, $\tau$ the timestamp, a the activity label, and $x_1$,...,$x_n$ any number of additional attributes. Labelling functions $eid$: e $\mapsto$ p, $time$: e $\mapsto \tau$, $act$: e $\mapsto$ a are included.

*Definition 3.2 (Trace):* A trace or process instance is a finite sequence of events t=$\langle$ $e_1$,...,$e_{|t|}\rangle$, with $|t|$ the number of events in that trace. The events are sequenced based on their timestamp $\tau$ such that $\forall$ $e_i$, $e_j$ $\in$ t: $i < j \rightarrow time(e_i) \leq time(e_j)$. The trace identifier is contained in its events: $\forall$ $e_i$, $e_j$ $\in$ t: $eid(e_i)$=$eid(e_j)$. This identifier can be retrieved through the labelling function $tid$: t $\mapsto$ $eid$ ($e_1$)=p. Two traces are equal if they have the same identifier: t=s $\iff$ $tid$(t)=$tid$(s).

*Definition 3.3 (Event Log):* An event log L is a set of traces. $|L|$ denotes its cardinality.

Sometimes, it is more useful to group traces that have the same control-flow pattern (disregarding their trace identifiers or timestamp). We refer to such traces as *distinct process instances*. An event log where traces are identified purely based on their sequence of activity labels is called a grouped event log.

*Definition 3.4 (Grouped Event Log):* In a grouped event log G, traces are considered equal if their sequence of events is equal: t=s $\iff |t| = |s| \wedge \forall i \in 1,...,|t| : act(t_i) = act(s_i))$. An element of a grouped event log is called a distinct process instance. The grouped event log G is a multiset of such distinct process instances, with $supp(G)$ denoting the number of distinct traces, or support of the multiset. $|G|$ denotes the cardinality, or its total size. The frequency of a distinct trace is the multiplicity of that trace in the grouped event log.

*Definition 3.5 (Trace Clustering):* A trace clustering C is a partition of an event log L: a set of nonempty subsets of L such that the union of all clusters is equal to the event log, and none of the clusters overlap: $\bigcup_{A \in C}$A = L $\wedge$ $\forall$ A, B $\in$ C: A$\cap$ B $\neq \emptyset \rightarrow$ A = B.

*Definition 3.6 (Process model):* A process model $PM$ is a graphical representation of a process.

*Definition 3.7 (Process discovery technique):* A process discovery technique PD is a function that maps an event log L onto a process model PM. $PD$: L $\mapsto$ PD(L)= PM.

*Definition 3.8 (Process model quality metric):* A process model quality metric $m$ is a function which returns a numeric value given an event log L and a process model $PM$. $m$: (L,$PM$) $\mapsto$ $m$(L,$PM$).

### B. Proposed Approach

The proposed approach, referred to as *TraCluSI*, is presented in Algorithm 1. It consists of roughly three phases: (1) creating an overclustering, (2) selecting representative super-instances for each cluster in the overclustering, and (3) clustering the super-instances. The algorithm requires two main inputs: a grouped event log and a desired number of clusters. Apart from that, a number of configuration options are available. More specifically, the number of super-instances, the super-instance assignment strategy, a process discovery technique, a process model quality metric and a cluster threshold value are needed. These will be discussed in more detail in Section III-C.[1]

In a first step, the grouped event log is featurized (line 3). This entails mapping each process instance onto a vector

[1]A prototype implementation of TraCluSI is available via `http://processmining.be/traclusi`.

**Algorithm 1** Algorithm TraCluSI (Trace Clustering using Super-Instances)

**Input:** $G$ := a Grouped Event Log, $k$:= the desired number of clusters

**Input:** Configuration: $n$:= the number of super-instances, $Strategy$:= a super-instance selection strategy, $PD$:= a process discovery technique, $m$:= a process model quality metric , $ctv$:= clustering threshold for the quality metric

**Output:** $\{C_j\}_{j=1}^{k}$ := An ordered set of clusters

1: $\{Super_i\}_{i=1}^{n}$:= $\emptyset$ % Initialize super-instances empty
2: $\{Sub_i\}_{i=1}^{n}$:= $\emptyset$ % Initialize sub-instances empty
3: $X$ := Featurize($G$) % $X$ is a clusterable dataset representing the traces
4: $\{O_i\}_{i=1}^{n}$:= K-Means($X$, $n$) % $O$ is an overclustering of $X$
5: **for** $i := (1 \rightarrow |n|)$ **do** % Assign super-instance for each cluster in the overclustering
6: **if** $Strategy$='Frequency' **then**
7: $Super_i$ := Most frequent distinct process instance in $O_i$
8: $Sub_i$ := Set of all other distinct process instances
9: **else if** $Strategy$='Centrality' **then**
10: $Super_i$ := Most central distinct process instance in $O_i$ % Closest distance to cluster centroid
11: $Sub_i$ := Set of all other distinct process instances
12: **end if**
13: **end for**
14: $\{SC_j\}_{j=1}^{k}$:= ActiTraC($S$, $k$, $PD$, $m$, $ctv$) % Cluster the super-instances in an active manner
15: **for** $j := (1 \rightarrow |k|)$ **do** % For each cluster
16: **for** $i := (1 \rightarrow |SC_j|)$ **do** % For each super-instance
17: $\{C_j\}$:= $C_j \cup Super_{SC_i} \cup Sub_{SC_i}$ % Add the super-instance and its corresponding sub-instances to the final clusters
18: **end for**
19: **end for**
20: **return** $C$

space model, such that these process instances can be clustered based on their position in this vector space. In theory, any featurization mapping traces onto features can be used. We propose using a combination of activity profiles [3], pairs and 3-grams [4]. Since the potential feature set size grows exponentially in the number of activity names, it can also be interesting to use a dimensionality reduction technique, such as Principal Component Analysis [18]. Next, this feature set is used to create an overclustering: a clustering algorithm is used to split the event log into a number of clusters equal to the number of super-instances $n$. For simplicity and performance, we propose using k-means clustering (line 4). This results in an overclustering, and concludes the first phase of our algorithm.

In a second phase, super-instances are selected to represent each of the $n$ clusters created in the previous step. We envision two approaches, with $Strategy$ as a switch: if 'Frequency' is chosen, the super-instances will be selected based on the frequency of the distinct process instances of all instances in each cluster (lines 5-8). If 'Centrality' is chosen as a strategy, the super-instance will be selected based on its distance to the centroid of the cluster: the super-instance that is closest to the centroid is selected (lines 9-11). Regardless of the chosen strategy, all non-selected instances, or 'sub-instances' are temporarily stored separately (lines 8 and 11).

In the third and final phase of the algorithm, the super-instances are clustered in a process model-driven manner. Here, we propose using ActiTraC [11]. The result is a clustered set of super-instances (line 14). Finally, to wrap up, all sub-instances are added to the clusters of their respective super-instance (lines 15-19).

### C. Configuration

In terms of configuration, a number of decisions have to be made. Clearly, the number of super-instances will influence the results. While using a higher number of super-instances could make the first phase of the algorithm faster, it will also raise the effort needed in the third phase. The number of super-instances also represents the trade-off between the (process) model-driven and distance-based paradigms: the lower the number of super-instances, the more the model-driven paradigm will influence the results.

The chosen super-instance selection strategy has a similar effect: relying on frequency is more in line with the workings of the *ActiTraC*-algorithm used in the third step, while choosing a centrality-driven strategy is more in line with the *k-means* approach of phase one.

Finally, the process discovery technique, process model quality metric and cluster threshold value are all used internally in the *ActiTraC*-algorithm. In its original implementation, the choices for *ActiTraC* are Heuristics miner [19], ICS-fitness [20] and 100%. Of course, alternative discovery techniques, process model quality metrics and quality thresholds could be applied as well.

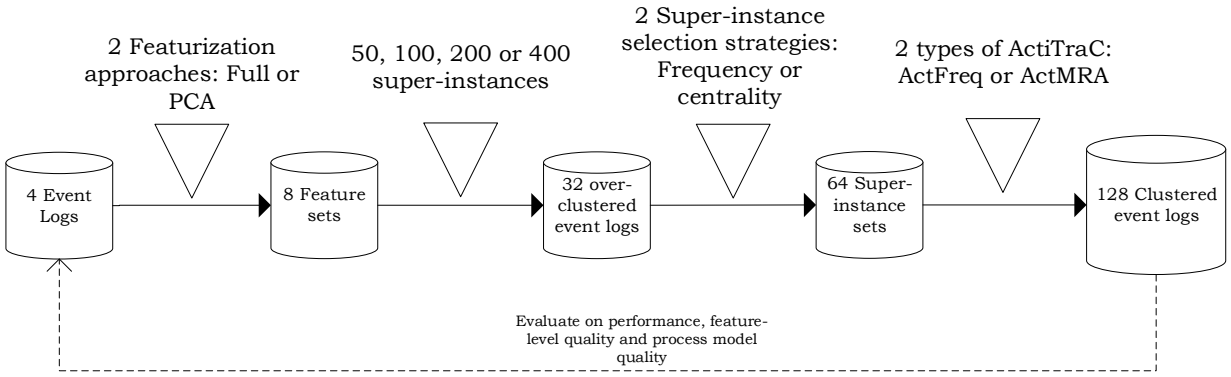



Fig. 1. Schematic overview of the experimental setup regarding the configuration of TraCluSI

## IV. Experimental evaluation

### A. Setup

In this section, the *TraCluSI*-algorithm will be evaluated on real-life event logs. The evaluation section is split into two parts, each with a distinct objective. On the one hand, the goal is to draw a distinction between configurations of the proposed approach. More specifically, the featurization approach, the number of super-instances, the super-instance selection strategy, and the configuration of the active clustering in step 3 will be tested. A schematic representation of this setup is presented in Figure 1. On the other hand, the results obtained by *TraCluSI* will be compared to a number of existing trace clustering techniques. For both parts of the experimental setup, the solutions will be evaluated according to three dimensions: internal consistency, process model quality, and computational performance.

**Internal consistency.** For this, a quality metric is used to determine how valid the clustering is. More specifically, the silhouette score will be used, which is a representation of within-cluster cohesion and between-cluster separation [21]. For each point $i$ in a cluster, let $A(i)$ represent the average distance to all other points in its cluster, and $B(i)$ the average distance to all points in a separate cluster, for which this average distance is minimal. The silhouette score of this point is then:

$$\begin{cases} 1 - \frac{A(i)}{B(i)} & | \ A(i) < B(i) \\ 0 & | \ A(i) = B(i) \\ \frac{B(i)}{A(i)} - 1 & | \ A(i) > B(i) \end{cases}$$

Finally, the silhouette score of the entire clustering is the average silhouette score over all traces. It is clear that a vector space is needed to determine distances between points. The features used for this are the same as in Phase 1 of *TraCluSI*: for each trace, binary vectors representing the presence of activities (1-grams), transitions (2-grams) and 3-grams. The distances are measured in a PCA-reduced vector space based on the 20 largest eigenvalues.

**Process model quality.** To measure this dimension, a process model will be discovered for each cluster using the *Fodina* process discovery technique [22]. The accuracy of each process model discovered per cluster is then measured using the F1-score as proposed in [23], where $p_B$ is a precision metric and $r_B$ is a recall metric:

$$F1_B = 2 * \frac{p_B * r_B}{p_B + r_B}$$

In this study, the recall metric we have chosen is behavioural recall $r_b$ [24], and the precision metric we use is $etc^p$ [25]. Finally, a weighted average F-score metric for the entire clustering solution is then calculated as follows, similar to the approach in [11], where $k$ is the number of clusters in $C$ and $n_i$ the number of traces in cluster $i$:

$$F1_C^{WA} = \frac{\sum_{i=1}^{k} n_i F1_i}{\sum_{i=1}^{k} n_i}$$

**Computational performance.** To measure performance of the algorithms, experiments are run on an Intel Core i7-4712HQ processing unit of a 16 GB Dell Precision M3800 workstation running Windows 10 Enterprise.

**Event logs.** All experiments are run on four event logs, the characteristics of which are presented in Table II. The most important aspect of these logs is the number of distinct process instances, since this represents the extent to which the data contains a wide array of distinct behaviour. All event logs have a high number of distinct process instances, ranging from 2502 to 6811. Since no true number of clusters is known for any of the event logs, the choice is made to require each tested clustering technique or configuration to divide the event logs into five clusters.[2]

**Alternative clustering techniques.** To compare the results of our novel clustering technique to existing approaches, four alternative clustering techniques are applied to each of the datasets. Two distance-based approaches are included: MRA, or Maximal Repeat Alphabet, which is a hierarchical clustering technique based on conserved patterns [5], and 3-gram, which is a hierarchical clustering technique based on 3-gram featurization [4]. Two model-driven clustering approaches

[2]Preprocessed versions of the event logs are available from `http://processmining.be/traclusi`.

are included: *ActFreq*, or frequency-based ActiTraC, and *ActMRA*, or distance-based ActiTraC [11]. Both techniques internally use Heuristics miner and ICS-fitness. The target value is set to 0.95 on all event logs except BPIC17, where the target fitness is set to 1.00 (since these techniques would otherwise return just a single cluster instead of 5 clusters). For comparison, the third phase of TraCluSI, in which the super-instances are clustered, will be configured in the same way, which we will refer to as TraCluSI-Freq and TraCluSI-MRA.

TABLE II
CHARACTERISTICS OF THE REAL-LIFE EVENT LOGS USED FOR THE EVALUATION: NUMBER OF PROCESS INSTANCES (**#PI**), DISTINCT PROCESS INSTANCES (**#DPI**), AVERAGE FREQUENCY OF EACH DISTINCT PROCESS INSTANCE ($\frac{\#PI}{DPI}$), NUMBER OF DIFFERENT EVENTS (**#EV**) AND AVERAGE NUMBER OF EVENTS PER PROCESS INSTANCE ($\frac{\#EV}{PI}$).

| **Log name** | **Source** | **#PI** | **#DPI** | $\frac{\#PI}{DPI}$ | **#EV** | $\frac{\#EV}{PI}$ |
|---|---|---|---|---|---|---|
| BPIC12 | [26] | 13087 | 4366 | 3.00 | 36 | 20.03 |
| BPIC15 | [27] | 5649 | 2502 | 2.26 | 29 | 16.36 |
| BPIC17 | [28] | 31509 | 6811 | 4.63 | 52 | 35.65 |
| BPIC18 | [29] | 8564 | 2896 | 2.96 | 75 | 43.08 |

### *B. Evaluating the configuration of TraCluSI*

**Internal consistency.** The results in terms of internal consistency are visualised in Figure 2. From the visualisation, it is clear that the main configuration impacting the silhouette score is the type of active clustering used in the third phase: TraCluSI-MRA consistently outperforms TraCluSI-Freq. With regards to super-instances, the preference goes to a lower number of super-instances, although the impact is not very large. In terms of selection strategy for the super-instances, there is no clear winner. In terms of featurization, the full featurization performs on par with the PCA-reduced set, except when combined with TraCluSI-Freq at higher numbers of super-instances. In summary, the best configuration of TraCluSI in terms of internal consistency, is one using PCA feature reduction, a low number of super-instances, and TraCluSI-MRA in the third phase, with no clear effect of the super-instance selection strategy.

**Process model quality.** Figure 3 contains the results of TraCluSI with regards to process model quality. Some trends appear, with varying degrees of severity. First, it appears that configuring TraCluSI with 100 super-instances performs best, although the best results when a frequency selection strategy is combined with TraCluSI-MRA are found at 400 super-instances. Additionally, the results of TraCluSI-MRA appear to be more stable than those of TraCluSI-Freq with regards to impact of the number of super-instances. Secondly, when using a frequency-based super-instance selection strategy, creating an overclustering based on the full feature set is superior. However, the inverse holds for the centrality-based super-instance selection strategy. There, the PCA-based overclustering scores better. Overall, configuring TraCluSI with a Centrality-based selection strategy, using a PCA-based featurization, and a low amount of super-instances seems preferable.

**Computational performance.** The evaluation of the computational complexity is split into two parts, based on the different phases of TraCluSI, see Algorithm 1. In the first phase, the traces are featurized and an overclustering is created based on the resulting vector space model. In this phase, the type of featurization and the number of super-instances are expected to have an impact. In the second phase, representative super-instances are selected based on the overclustering. This step can be done based on frequency or based on centrality, but it is linear in terms of the number of distinct process instances, and is not expected to add to the runtime significantly. Finally, in the third phase, the chosen super-instances are clustered in a model-driven manner. Here, we are expecting an effect of the number of super-instances (the more super-instances to cluster, the slower), an effect of the active clustering strategy (TraCluSI-Freq or TraCluSI-MRA).

The results of the first phase are summarized in Table III. Three main takeaways are apparent from this table: first, creating the overclustering based on the 20 most important principal components is decidedly more efficient than using the full feature set: about 21 times quicker on average. Secondly, when using the dimensionally reduced features, doubling the number of super-instances roughly increases the expected runtime with 50%. A similar ratio is found when using the full feature set. Finally, these tendencies hold over both average numbers and medians.

TABLE III
EFFECT OF NUMBER OF FEATURIZATION AND NUMBER OF SUPER-INSTANCES ON TIME NEEDED TO CREATE OVERCLUSTERING (IN SECONDS). AVERAGE AND MEDIAN OVER 4 EVENT LOGS.

| Featurization | # of SIs | AVG(Runtime) | MEDIAN(Runtime) |
|---|---|---|---|
| Full | 50 | 20.12s | 13.58s |
| Full | 100 | 31.70s | 21.21s |
| Full | 200 | 45.43s | 23.92s |
| Full | 400 | 81.04s | 43.13s |
| **PCA** | **50** | **0.81s** | **0.53s** |
| PCA | 100 | 1.34s | 0.83s |
| PCA | 200 | 2.37s | 1.24s |
| PCA | 400 | 3.74s | 2.11s |

The results on the second and third phase are presented visually in Figure 4. The results shown are based on the PCA-reduced features, given the results in phase one. The most obvious difference is caused by the chosen active clustering strategy: TraCluSI-Freq consistently outperforms TraCluSI-MRA. Furthermore, the number of super-instances influences

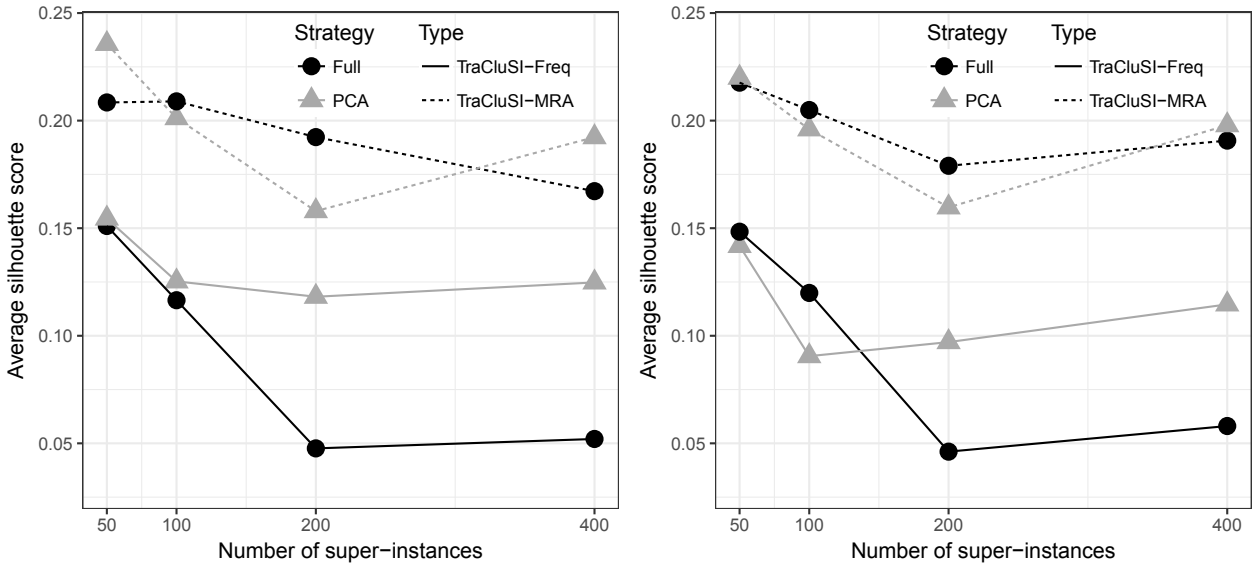



Fig. 2. Average silhouette scores for different configurations of TraCluSI over four event logs, with Frequency selection strategy on the left, and Centrality on the right.

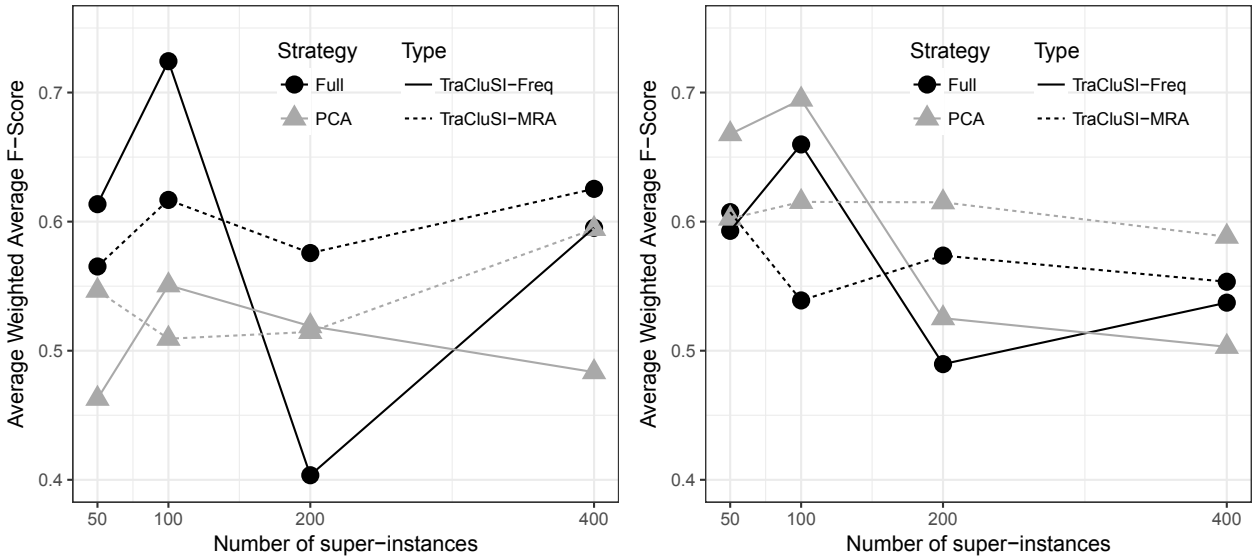



Fig. 3. Average F-Score results for different configurations of TraCluSI over four event logs, with Frequency selection strategy on the left, and Centrality on the right.

the runtime, with a higher number of super-instances leading to higher computational time needed. This effect is more notable for TraCluSI-MRA than for TraCluSI-Freq, however. Finally, as expected, the chosen super-instance selection strategy does not influence the results in terms of computational complexity.

To summarize, the best configuration of TraCluSI in terms of expected computational performance is a TraCluSI-Freq configuration, a low number of super-instances and PCA-reduced featurization.

### C. Comparison to existing clustering techniques

**Internal consistency.** The results in terms of internal consistency are presented in Table IV. For the existing techniques, these are averages over four event logs. For TraCluSI, these are averages over the event logs, featurization strategies, number of super-instances and super-instance selection strategies, grouped by the active clustering strategy used in phase 3. For a more detailed view on the results of TraCluSI, see Figure 2. TraCluSI-MRA outperforms the best alternative, pure *ActMRA*, when measured by mean, with a tie based on median.

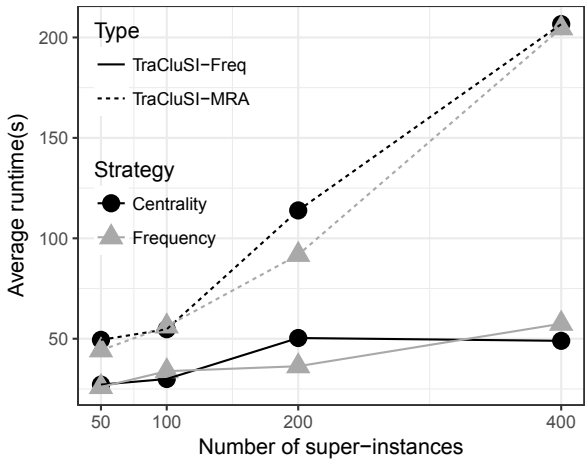



Fig. 4. Average runtime in seconds of the second and third phase of the algorithm, after PCA featurization in phase one, for varying levels of super-instances, selection strategies and active clustering techniques

TABLE IV
RESULTS OF SILHOUETTE SCORES OF EXISTING TRACE CLUSTERING TECHNIQUES. AVERAGES AND MEDIANS OVER FOUR EVENT LOGS. TRACLUSI-FREQ/-MRA REPRESENT AVERAGES OVER ALL CONFIGURATIONS OF TRACLUSI-FREQ/-MRA.

| Algorithm | AVG(Silhouette) | MEDIAN(Silhouette) |
|---|---|---|
| **TraCluSI-MRA** | **0.20** | 0.15 |
| ActMRA | 0.13 | 0.15 |
| TraCluSI-Freq | 0.11 | 0.09 |
| 3-gram | 0.09 | 0.08 |
| ActFreq | 0.02 | 0.02 |
| MRA | -0.01 | -0.04 |

**Process model quality.** Table V contains the results in terms of weighted average F-scores for the existing trace clustering techniques, averaged over the four event logs used. As a comparison, two of the better performing configurations of TraCluSI are included as well. For a more general overview of the performance of TraCluSI over all configurations, we refer to Figure 3. A couple of remarks can be made from these results. First, observe how all existing trace clustering techniques perform somewhat similar. This is surprising, since only *ActFreq* and *ActMRA* are model-driven trace clustering techniques. Secondly, most discovered process models score well on recall, which means that the F-score is largely dominated by the precision results. Given the large event logs, the five resulting clusters are still large event logs themselves, so it makes sense that precision of discovered process models for each cluster is not outstanding. Nonetheless, it is clear that some of the better configurations of TraCluSI, as included in the table, are able to perform better than the existing traces clustering techniques.

**Computational performance.** The results in terms of computational complexity of the four existing trace clustering techniques are presented in Table VI. The worst (full feature set used in phase 1, 400 super-instances, frequency selection in phase 2, and *ActMRA* in phase 3) and best configurations (PCA-reduced feature set, 50 super-instances, frequency selection and *ActFreq* in phase 3) are included as well. From Table VI, it is apparent that the best-case configuration of TraCluSI outperforms the competition. The worst-case configuration scores better than the fastest existing clustering technique (*3-gram*) as well, but only when looking at averages. *3-gram* is faster based on median performance over the four datasets. Finally, notice the severe computational time needed by the two process-model aware techniques (*ActFreq* and *ActMRA*). The former takes 35 minutes on average to cluster one of the event logs, and the latter over 17 minutes.

TABLE V
COMPARISON OF TRACLUSI RESULTS TO EXISTING CLUSTERING TECHNIQUES IN TERMS OF F-SCORE. TRACLUSI-FREQ IS PCA FEATURIZATION, 100 SUPER-INSTANCES, CENTRALITY SELECTION STRATEGY, AND APPLYING ACTFREQ IN THIRD PHASE. TRACLUSI-MRA IS CONFIGURED THE SAME, EXCEPT WITH ACTMRA IN THE THIRD PHASE. AVERAGES AND MEDIANS OVER 4 EVENT LOGS.

| Algorithm | AVG(F-Score) | MEDIAN(F-Score) |
|---|---|---|
| **TraCluSI-Freq** | **0.69** | **0.69** |
| TraCluSI-MRA | 0.62 | 0.60 |
| ActMRA | 0.61 | 0.57 |
| 3-gram | 0.57 | 0.57 |
| MRA | 0.56 | 0.56 |
| ActFreq | 0.54 | 0.52 |

TABLE VI
COMPARISON OF BEST AND WORST TRACLUSI RESULTS TO EXISTING CLUSTERING TECHNIQUES IN TERMS OF RUNTIME. WORST IS FULL FEATURES, 400 SUPER-INSTANCES, FREQUENCY SELECTION STRATEGY AND ACTMRA IN THIRD PHASE. BEST IS PCA, 50 SUPER-INSTANCES, FREQUENCY SELECTION STRATEGY AND ACTFREQ IN PHASE 3. AVERAGES AND MEDIAN OVER 4 EVENT LOGS.

| Algorithm | AVG(Runtime) | MEDIAN(Runtime) |
|---|---|---|
| **TraCluSI-Best** | **26.795s** | **18.8s** |
| TraCluSI-Worst | 287.82s | 171.94s |
| 3-gram | 366.03s | 159.53s |
| MRA | 684.97s | 538.07s |
| ActMRA | 1056.56s | 1081.74s |
| ActFreq | 2101.60s | 1884.58s |

## V. CONCLUSIONS

This paper presents a novel trace clustering technique, TraCluSI, that is capable of clustering traces in a mixed-

paradigm manner. The combination between both paradigms is based on the concept of super-instances, or smaller subsets of traces chosen to represent the event log. These super-instances are produced by overclustering the dataset using a fast, distance-driven clustering technique (k-means), which guarantees that highly similar traces are grouped together, while the grouping of the super-instances to obtain the final clustering solution is performed entirely model-driven so as to optimize the process model quality of the models underlying the produced clusters. The end result is a clustering technique that performs well in terms of scalability, internal consistency and discovered process model quality, compared to existing trace clustering techniques. Furthermore, the optimal configuration of the algorithm is studied. This leads to the conclusion that a lower number of super-instances is likely desirable and that *ActFreq*-based clustering of the super-instances is best for scalability and process-model quality, but *ActMRA*-based clustering performs better in terms of internal consistency. PCA-based featurization positively impacts runtime, and it combines better with centrality-based super-instance selection than frequency-based super-instance selection with regards to process model quality.

Considering future work, the most interesting avenue is investigating extensions and enhancements of TraCluSI. Firstly, changing the frequency-based super-instance selection strategy to incorporate a top percentage of frequent traces instead of just the most frequent trace could be useful. Secondly, the super-instance selection and model-driven clustering of phases 2 and 3 could be replaced by a model-driven merging of clusters from the overclustering, based on similarity of process models discovered for each cluster. Finally, TraCluSI is ideally suited for incorporation of expert knowledge. Such a semi-supervised extension could apply expert knowledge to the featurization in phase 1, to the super-instance selection in phase 2, or cluster the super-instances in a semi-supervised manner in phase 3.

## REFERENCES


[1] T. Thaler, S. F. Ternis, P. Fettke, and P. Loos, "A comparative analysis of process instance cluster techniques." *Wirtschaftsinformatik*, vol. 2015, pp. 423–437, 2015.

[2] G. Greco, A. Guzzo, L. Pontieri, and D. Saccà, "Discovering expressive process models by clustering log traces," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 8, pp. 1010–1027, 2006.

[3] M. Song, C. Günther, and W. M. van der Aalst, "Trace Clustering in Business Process Mining," in *Bus. Process Manag. Work.*, vol. 17. Springer, 2009, pp. 109–120.

[4] R. P. J. C. Bose and W. M. P. van Der Aalst, "Context Aware Trace Clustering: Towards Improving Process Mining Results." *Sdm*, pp. 401–412, 2009.

[5] ——, "Trace clustering based on conserved patterns: Towards achieving better process models," in *Lect. Notes Bus. Inf. Process.*, vol. 43 LNBIP, 2010, pp. 170–181.

[6] P. Delias, M. Doumpos, E. Grigoroudis, P. Manolitzas, and N. Matsatsinis, "Supporting healthcare management decisions via robust clustering of event logs," *Knowledge-Based Syst.*, vol. 84, pp. 203–213, 2015.

[7] A. Appice and D. Malerba, "A co-training strategy for multiple view clustering in process mining," *IEEE Transactions on Services Computing*, vol. PP, no. 99, pp. 1–1, 2015.

[8] J. Evermann, T. Thaler, and P. Fettke, "Clustering traces using sequence alignment," in *Business Process Management Workshops: BPM 2015, 13th International Workshops, Innsbruck, Austria, August 31 – September 3, 2015, Revised Papers*, M. Reichert and A. H. Reijers, Eds. Springer International Publishing, 2016, pp. 179–190.

[9] D. R. Ferreira, M. Zacarias, M. Malheiros, and P. Ferreira, "Approaching process mining with sequence clustering: Experiments and findings," in *BPM*, 2007, pp. 360–374.

[10] F. Folino, G. Greco, A. Guzzo, and L. Pontieri, "Editorial: Mining Usage Scenarios in Business Processes: Outlier-aware Discovery and Run-time Prediction," *Data Knowl. Eng.*, 2011.

[11] J. De Weerdt, S. vanden Broucke, J. Vanthienen, and B. Baesens, "Active trace clustering for improved process discovery," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 12, pp. 2708–2720, 2013.

[12] T. Chatain, J. Carmona, and B. Van Dongen, "Alignment-based trace clustering," in *International Conference on Conceptual Modeling*. Springer, 2017, pp. 295–308.

[13] P. De Koninck, K. Nelissen, B. Baesens, S. vanden Broucke, M. Snoeck, and J. De Weerdt, "An approach for incorporating expert knowledge in trace clustering," in *Advanced Information Systems Engineering: 29th International Conference, CAiSE 2017, Essen, Germany, June 12-16, 2017, Proceedings*, E. Dubois and K. Pohl, Eds. Springer International Publishing, 2017, pp. 561–576.

[14] A. K. Jain and R. C. Dubes, *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[15] I. Cadez, D. Heckerman, C. Meek, P. Smyth, and S. White, "Model-based clustering and visualization of navigation patterns on a web site," *Data Mining and Knowledge Discovery*, vol. 7, no. 4, pp. 399–424, 2003.

[16] S. Guha, R. Rastogi, and K. Shim, "Cure: an efficient clustering algorithm for large databases," in *ACM Sigmod Record*, vol. 27, no. 2. ACM, 1998, pp. 73–84.

[17] T. Van Craenendonck, S. Dumancic, and H. Blockeel, "Cobra: A fast and simple method for active clustering with pairwise constraints," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, 2017, pp. 2871–2877.

[18] M. Song, H. Yang, S. H. Siadat, and M. Pechenizkiy, "A comparative study of dimensionality reduction techniques to enhance trace clustering performances," *Expert Syst. Appl.*, vol. 40, pp. 3722–3737, 2013.

[19] A. J. M. M. Weijters, W. M. P. van der Aalst, and A. K. A. de Medeiros, "Process mining with the heuristicsminer algorithm," TU Eindhoven, BETA Working Paper Series 166, 2006.

[20] A. J. M. M. Weijters and W. M. P. van der Aalst, "Rediscovering Workflow Models from Event-Based Data using Little Thumb," *Integr. Comput. Eng.*, vol. 10, pp. 151–162, 2003.

[21] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53 – 65, 1987.

[22] S. K. vanden Broucke and J. D. Weerdt, "Fodina: A robust and flexible heuristic process discovery technique," *Decision Support Systems*, vol. 100, pp. 109 – 118, 2017, smart Business Process Management.

[23] J. De Weerdt, M. De Backer, J. Vanthienen, and B. Baesens, "A robust f-measure for evaluating discovered process models," in *Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on*. IEEE, 2011, pp. 148–155.

[24] S. Goedertier, D. Martens, J. Vanthienen, and B. Baesens, "Robust Process Discovery with Artificial Negative Events," *J. Mach. Learn. Res.*, vol. 10, pp. 1305—-1340, 2009.

[25] J. Muñoz-Gama and J. Carmona, "A fresh look at precision in process conformance," in *Business Process Management: 8th International Conference, BPM 2010, Hoboken, NJ, USA, September 13-16, 2010. Proceedings*, R. Hull, J. Mendling, and S. Tai, Eds. Springer Berlin Heidelberg, 2010, pp. 211–226.

[26] B. Van Dongen, "Bpi challenge 2012," 2012. [Online]. Available: https://data.4tu.nl/repository/uuid:3926db30-f712-4394-aebc-75976070e91f

[27] ——, "Bpi challenge 2015," 2015. [Online]. Available: https://data.4tu.nl/repository/uuid:31a308ef-c844-48da-948c-305d167a0ec1

[28] ——, "Bpi challenge 2017," 2017. [Online]. Available: https://data.4tu.nl/repository/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b

[29] B. Van Dongen and F. Borchert, "Bpi challenge 2018," 2018. [Online]. Available: https://data.4tu.nl/repository/uuid:3301445f-95e8-4ff0-98a4-901f1f204972

# Directly Follows-Based Process Mining: Exploration & a Case Study

Sander J.J. Leemans, Erik Poppe, Moe T. Wynn
Queensland University of Technology
Brisbane, Australia
{s.leemans, e.poppe, m.wynn}@qut.edu.au

***Abstract*—Many organisations now seek to analyse and improve their processes using event logs from various IT systems supporting their operations. Process mining aims to obtain insights from such process data, using process discovery, conformance checking and performance measures. While many commercial process mining tools feature user-friendly directly follows-based process maps, they typically do not offer a way to assess the quality of the model, leaving users with potentially unreliable insights, which could lead to the wrong conclusion being drawn from these insights. In contrast, academic tools typically provide verifiable results, but are often difficult to use and understand for stakeholders, sometimes overgeneralising behaviour to fit more extensive process model formalisms. In this paper, we bridge this well-known gap between commercial and academic tools by combining sound process discovery, conformance checking and performance capabilities with user-friendly directly follows-based process models. We implemented these techniques in a new process mining tool and applied them to analyse several business processes in a Queensland Government department. We discovered sound directly follows-based process models from their logs, compared them with prescribed models and analysed the performance of these processes. In particular, our conformance checking techniques allowed to pinpoint deviations between prescribed processes and actual recorded behaviour. The outcomes of this case study are now being used to document, review, improve and automate processes.**

## I. Introduction

Organisations store large amounts of data nowadays, recorded from process executions in workflow systems, web applications or tracking technologies. From such transaction data (e.g., purchase order, customer journey, insurance claim, travel booking), an event log can be generated that captures the process steps executed for a trace. Process mining techniques and tools aim to derive meaningful information from such recorded event logs [1].

A typical first step in a process mining study is to automatically discover a process model from an event log, that shows the process steps (activities) that were performed and their order. Such a process model can then be used to compute performance measures, and identify bottlenecks and staff workload. Thus, it is imperative that a process model reflects the 'reality' found in an event log as much as it can. In many cases, process discovery techniques make trade-offs between the four quality dimensions of fitness, precision, simplicity and generalisation. One way to measure the deviations between the log and a discovered model is by performing conformance checking analysis.

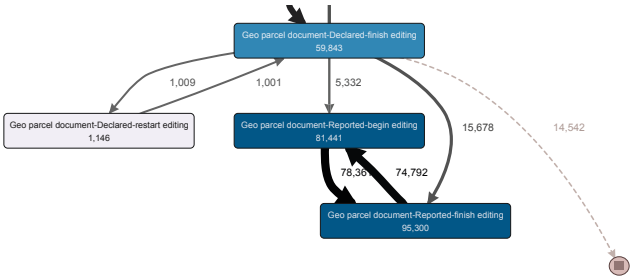



Fig. 1. Excerpt of a directly follows-based model.

Academic process mining tools typically use modelling formalisms with well-defined semantics, such as Petri nets, BPMN models and process trees, and support complex constructs such as concurrency, interleaving and inclusive choices. These semantics and constructs make them suitable for complex processes, but reduce simplicity [2]. For rather simple business processes, traditional process discovery techniques might need to overgeneralise in order to fit their representational bias [3] or return models with deadlocks or other anomalies (unsound models) [4].

In contrast, many commercial process mining tools exist that typically discover process models in the form of directly follows-based process maps. That is, their visualisations show which activities can follow one another directly. These commercial tools also project time and frequency based performance measures such as execution times, wait times and average or maximum activity counts on these maps. Due to the simplicity of these maps and the interactive filtering options offered by these commercial tools, these tools have captured a large market share[1].

Unfortunately, those same characteristics can make the interpretation of these maps and the related performance measures inaccurate. Some of these limitations are illustrated in Figure 1. Figure 1 shows an example of a directly follows-based model that has been filtered: cases start at the top of the model and progress down through activities to the circle at the bottom. As a result of filtering, the semantics of this model are not clear, as there is no way to reach the end circle from the two bottom-most blue activities, and the numbers, indicating how often each activity and edge has been executed, seem to be inaccurate, as 29000 cases start and only 14500 end.

[1]See https://www.gartner.com/doc/3870291/market-guide-process-mining

978-1-7281-0919-0/19/$31.00 ©2019 IEEE
DOI 10.1109/ICPM.2019.00015

Without a clear understanding of the effects of filtering on such numbers, it is easy to draw the wrong conclusions from filtered models.

Conformance checking analysis on directly follows-based models can benefit the stakeholders in evaluating the quality of discovered models. However, although conformance checking has been part of academic process mining tools for some time, support in commercial tools is rather sparse to date. Furthermore, none of the mentioned commercial tools allows the manual import of a process map, for instance representing an authoritative model, against which the behaviour in the log can be compared.

This paper bridges the gap between current directly follows-based maps produced by commercial tools and well-founded but complex process models generated by academic tools. We show how a directly follows-based process mining tool can support process discovery with well-defined semantics, conformance checking and reliable performance measures. We present a case study where this tool has been used to analyse multiple processes from a Queensland Government department. It is our hope that these findings will inspire commercial tools to adopt a similar notion in future release of these tools.

In particular, this paper contributes:

- A definition of *directly follows models* (DFMs), their semantics and their soundness;
- The application of conformance checking to DFMs;
- A new way to discover DFMs by filtering infrequent behaviour while guaranteeing soundness and a minimum fitness level;
- A description of reliable performance measures for DFMs;
- A user-friendly implementation (tool) that includes process discovery, conformance checking and performance measures for DFMs; and
- A case study in which we applied conformance checking and performance measures to (hand-made and discovered) directly follows models and event logs from a Queensland Government department.

We discuss related work in Section II and introduce basic concepts in Section III. In Section IV we discuss how DFMs can be discovered from event logs, in Section V how standard conformance checking techniques can be applied and their results visualised, and in Section VI how performance measures can be computed reliably. In Section VII, we describe the case study performed at the Queensland Government, and we conclude the paper in Section VIII.

## II. Related Work

This section briefly summarises recent work on process model formalisms, process mining tools, types of process mining analysis (such as conformance checking and performance analysis) and case studies.

*a) Process Model Formalisms:* In [2], three drivers for directly follows-based models in commercial tools are given: simplicity (Petri nets or BPMN models with too many gateways are considered un-interpretable), vagueness (ability to express relationships that cannot be interpreted in a precise manner), and scalability (ability to handle logs with millions of events interactively).

The most basic process modelling notation is a transition system which consists of states and transitions between states that correspond to activities being executed. All atomic (i.e. steps do not take time) business processes, such as the higher-level Petri nets [5], some BPMN models [6], process trees [3] and Declare models [7], can be expressed as transition systems.

Directly follows models (DFMs) are variants of transitions systems and lie in between transition systems and higher-level languages: while in a transition system the execution of an activity is atomic, in DFMs this takes time. However, in contrast to the higher-level languages, DFMs do not support concurrency. Furthermore, there is a difference in focus between transition systems and DFMs: while DFMs focus on the ordering of activities, transition systems focus on the state of a process.

BPMN is a standard that provides both graphical representations and execution semantics for processes. BPMN supports many advanced behaviour features such as concurrency, event handling and hierarchy and has found significant adoption by industry, but its complexity means that it requires training for stakeholders to be correctly understood.

While in [2] a new hybrid process modelling concept is proposed that combines formal (Petri nets) with informal ("sure" and "unsure") parts, our approach aims to illustrate the possibilities for directly follows-based models as used by commercial tools. That is, we keep the simplicity and address the vagueness by providing formal semantics for DFMs, defining soundness and provide reliable performance measures, although scalability might suffer from our use of alignments (see Section III).

*b) Process Mining Techniques and Tools:* In this paper, we consider three types of process mining techniques: process discovery (discovering a process model from an event log), conformance checking (comparing a process model with an event log) and performance measuring.

Discovering directly-follows models, while simple, will often produce large models, so many directly follows-based discovery tools attempt to reduce complexity through abstraction and aggregation. For instance, Heuristic Miner [8] removes behavioural relations based on occurrence frequency. Fuzzy Miner [9] similarly removes directly follows relations based on several heuristics, but supports abstraction by grouping activities in addition. Both approaches, however, can produce unsound models, as they remove behaviour that could be necessary to reach the end state of the process, thus potentially leaving the semantics of the produced models unclear.

Discovery algorithms typically generate process models of higher-level process model formalisms, containing high-level process models with constructs such as concurrency, inclusive choices and interleaving. However, some algorithms

may return unsound models ($\alpha$ [1], Split Miner [4], BPMN miner [10], Fodina [11]), require considerable computations times (Evolutionary Tree Miner [12]) or overgeneralise behaviour (Inductive Miner [3]). Furthermore, some techniques generalise transition systems to higher-level process models [13], [14].

Commercial process mining tools, such as Fluxicon Disco (FD) [15], Celonis Process Explorer (CPE), ProcessGold (PG) and myInvenio (MI) [2], typically generate directly follows-based "models". Also the academic Fuzzy Miner (FM) [9] and other approaches [16] focus on directly follows-based models. In order to reduce the complexity of models, these tools seem to filter based on edges (FD, FM, CPE) and/or traces (CPE, PG), which may lead to models with soundness issues (see Section IV). The models are simplistic and do not capture concurrency. However, they can be interpreted easily by domain experts and are thus more easily accepted by users.

In this paper, we define soundness for DFMs and introduce a discovery technique that guarantees DFM-soundness.

*c) Conformance Checking and Enhancement Techniques:* Conformance checking techniques provide ways to compare behaviour in event logs with behaviour described by process models, which can be either discovered or created manually. As the expressiveness of a process modelling language has a direct effect on the simplicity of models and analysis capabilities, conformance checking techniques are mostly based on process models represented as a Petri net or process tree. Alignments [17], [7], explained in Section III, can be used to identify deviating behaviour. Existing work on conformance checking using directly follows process models has been limited to rule based, outlier (low-frequency behaviour) based or graph-based approaches (as in MI), in which the user has to find visual differences across multiple models. CPE supports conformance checking, however in a disjoint component: it supports a different syntax (BPMN), discovery technique and concepts, and might yield unintuitive results[3].

Process enhancement techniques derive additional information such as performance data using a process model and an aligned event log, see for instance [18], [19], [20]. In Section VI, we will show that existing tools might yield unintuitive performance results.

Our approach highlights deviations between log and model based on alignments and provides reliable performance metrics by projecting alignments onto directly-follows models.

*d) Case Studies:* Process mining techniques have been used in a number of process mining case studies in various domains such as insurance [18], healthcare [21] education [22] and government services [23]. In a number of such case studies, comparing the performance of different process variants is one of the key activities. When such comparative performance analysis is conducted on directly follows maps without careful attention paid to the abstractions made to the underlying process models, the results are not comparable and reliable. Our approach realigns the log with the abstracted model, making the performance measures reliable and comparable.

[2]See https://www.celonis.com/, https://processgold.com/en/, https://www.my-invenio.com/

[3]See https://community.celonis.com/t/unexpected-results-in-conformance-checking/1183.

## III. Preliminaries

*a) Event Logs:* An *event log* is a collection of *traces*, each consisting of the process steps to be executed. We refer to the steps as *activities* and their execution in a trace as *activity instances*. Each activity instance might traverse a certain life cycle. In this paper, we assume a simple cycle of an optional start and a completion. That is, each trace is a sequence of *events*, which represent the life cycle transitions of activity instances. For instance, $[\langle a_s, a_c, b_c, c_s, c_c\rangle, \langle a_s, a_c, c_s, c_c, b_c\rangle]$ is an event log consisting of two traces: in the first trace, first an instance of the activity $a$ started, then it completed, then an instance of $b$ completed (the start of this instance of $b$ was not recorded) and finally $c$ was started and completed. We assume that in each trace, each start event has a corresponding completion event, but that completion events do not need a corresponding start event. We chose this asymmetry as the event logs we encountered in practice typically contain completion events but rarely contain start events [24]. Similarly, a *language* is a set of traces, and the empty trace is denoted by $\epsilon$.

*b) Petri Nets:* A Petri net is a graph consisting of *places* that can contain *tokens*, the presence of tokens determines the state of the net, as well as *transitions* which change the state of the net by consuming and producing tokens from/to connected places [5] and emitting an associated activity by *firing*. In a *workflow* (Petri) net, there is a single place without incoming (source) and a single place without outgoing (sink) transitions, and each place and transition is on a directed path between these two places. A workflow net is *sound* if all its transitions can be fired, and from each reachable state the end state (having the only token in the net in the sink) is reachable [1].

*c) Alignments:* An *alignment* is a combination of a trace and a path through the model that produces that trace. If the trace cannot be produced by the model, then a flawless combination is not possible. Two types of deviations exist: skipping events in the trace (a *log move*) or skipping activities in the model (a *model move*) [17]. An alignment is *optimal* if it has the least cost of all possible alignments, which is typically chosen to be the number of deviations.

For instance, consider the DFM shown in Figure 2 and the trace $\langle a_s, a_c, c_s, c_c\rangle$. Then an optimal alignment is:

| log | $a_s$ | $a_c$ | $c_s$ | $c_c$ | - |
|---|---|---|---|---|---|
| model | $a_s$ | $a_c$ | - | - | $b_c$ |

The first two moves are *synchronous*, while the third and fourth moves are log moves on $c$, and the last move is a model move on $b$.

*d) Directly Follows Models:* Syntactically, a *directly follows model* (DFM) is a directed graph in which the nodes are annotated with either an activity, $start$ or $end$:

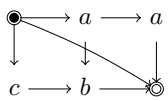



Fig. 2. Example of a directly follows model (DFM).

**Definition 1** (Directly follows model - syntax)**.** *Given an alphabet $\Sigma$ such that $start \notin \Sigma$ and $end \notin \Sigma$, a* directly follows model *is a directed graph $(N, E)$, such that $N : \Sigma \cup \{start, end\}$ is a set of nodes and $E : N \times N$ is a set of edges. Additionally, $start$ has no incoming and $end$ has no outgoing edges: $\neg\exists_n(n, start) \in E$ and $\neg\exists_n(end, n) \in E$.*

Semantically, the language of a DFM consists of all traces that can be obtained by starting in the $start$-node and walking along the edges, adding events for each activity encountered, to the $end$-node. Similar to event logs, a start event is optional while a completion event is mandatory for each activity to ensure the activity is actually executed.

**Definition 2** (Directly follows model - semantics)**.** *Let $D = (N, E)$ be a DFM. Then, the language of $D$ ($\mathcal{L}(D)$) is $\{\langle a_{1s}, a_{1c}, \dots a_{ns}, a_{nc}\rangle \mid a_1 \dots a_n \in N \wedge (start, a_1) \in E \wedge \forall_{1 \le i < n}(a_i, a_{i+1}) \in E \wedge (a_n, end) \in E\} \cup \{\epsilon \mid (start, end) \in E\}$, where all the $_s$-events are optional.*

For instance, Figure 2 shows a graphical representation of a DFM. The language of this DFM is $\{\epsilon, \langle a_s, a_c, a_s, a_c\rangle, \langle a_s, a_c, b_s, b_c\rangle, \langle c_s, c_c, b_s, b_c\rangle\}$.

Notice that the DFM formalism supports both the empty trace ($\{\epsilon\}$) as well as the empty language ($\emptyset$). However, concurrency is not supported by DFMs. For instance, there is no DFM that can represent the trace $\langle a_s, b_s, b_c, a_c\rangle$, as DFMs inherently cannot represent concurrency.

## IV. DF-Process Discovery

Many commercial and some academic tools offer DFM-discovery, however they might produce unsound models. In this section, we first define soundness for DFMs, after which we introduce a discovery method that results in sound DFMs and offers a user-chooseable fitness guarantee.

### A. Soundness

As stated in [1], unsound workflow nets might be useful for limited manual human analysis, but issues such as unexecutable activities, livelocks and deadlocks challenge both human and in particular automatic analysis.

In existing directly follows-based tools, a DFM is constructed, after which edges are filtered out based on a user-chosen parameter to reduce the complexity of the model. However, this may lead to soundness issues, as illustrated in Figure 1, which shows a DFM-like model. In this model, from the two middle-bottom activities, the end state cannot be reached. Despite the visual suggestion, these activities cannot be part of a trace that starts and ends properly, so the language of the model remains unclear.

We define soundness for DFMs, and show the connection between soundness of DFMs and soundness of workflow nets.

**Definition 3** (DFM soundness)**.** *Let $(N, E)$ be a DFM. Then, the DFM is* sound *if every node $\in N$ is on a path from $start$ to $end$:*

$$\forall_{x \in N} \exists_{a_1 \dots a_n \in N} a_1 = start \wedge a_n = end \wedge \exists a_j = x \wedge \forall_{1 \le i < n}(a_i, a_{i+1}) \in E$$

As a consequence of this definition, the DFM without any edges is not sound, as the $start$ and $end$ nodes are not on a path as requested by the definition. This implies that a sound DFM with the empty language does not exist, which is analogous to workflow nets: a workflow net with the empty language cannot be sound.

To further establish the link between sound DFMs and sound workflow nets, we show that Definition 3 is sufficient to translate a DFM to a sound workflow net:

**Lemma 1.** *A sound DFM can be translated to a language-equivalent sound workflow net.*

We will prove this lemma in Section V-A.

### B. Trace-Based DFM Discovery

In order to discover a DFM from an event log, as DFMs do not support concurrency, first all start events need to be removed from the traces. Then, a DFM can be obtained from an event log straightforwardly by sequentially following the events for traces in the log and adding arcs to the DFM accordingly. However, for a typical real-life process, this yields complex and unreadable DFMs.

Therefore, existing tools filter the DFM, either by removing infrequent *traces* before discovery or infrequent *edges* after discovery (next to removing activities). Tools that remove edges do so using certain criteria, typically removing the least-occurring edges. However, this strategy might yield unsound DFMs, as shown in Figure 1, which, when translated to Petri nets (Section V-A), might not result in sound workflow nets.

Tools that aim to filter infrequent behaviour by removing traces from the event log have to figure out which traces represent infrequent behaviour and should be removed. In complex processes, many or all traces might occur rarely, thus one needs to find the infrequent behaviour to decide which traces to remove [25]. *Trace-based DFM discovery* combines these two approaches:

1) First, a DFM is constructed by adding, for each trace, nodes and edges accordingly, while keeping track of how often each edge occurs in the log;
2) Second, the least frequent edges of the DFM are selected;
3) Third, all traces that use the selected edge are removed from the event log;
4) This procedure is repeated until a user-chosen threshold of removed traces is about to be exceeded. That is, the user-chosen threshold indicates how many traces will at most be filtered from the DFM.

Consequently, the DFM returned fits at least *threshold* of the traces in the event log (if only completion events are

considered), which is a guarantee not offered by other process discovery techniques.

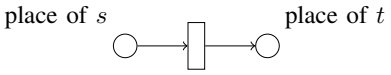
## V. DF-Conformance Checking

Conformance checking techniques compare an event log with a process model, which can be constructed either manually (to indicate prescriptive behaviour) or by process discovery techniques. The state-of-the-art technique to perform detailed conformance checking is to compute alignments on sound Petri nets (see Section III). In this section, we describe how alignments can be applied to DFMs and how the results can be visualised on DFMs, in order to show them in DF-based [commercial] tools. That is, we first describe how DFMs can be translated to Petri nets, and that sound DFMs yield sound workflow nets. Second, we introduce directly follows-specific alignment visualisation concepts.

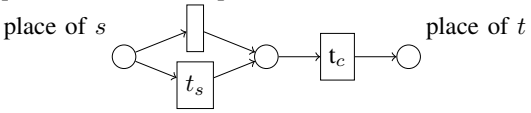
### A. Translation to Petri Nets

In Definition 2, we defined the language semantics of DFMs. To enable the easy use of DFMs in other process mining tools, we provide a translation to standard Petri (workflow) nets. In this translation, each transition is modelled as a combination of a start and a completion event, such that performance measures can be computed. For a DFM $(N, E)$, the translated Petri net consists of:

- A place for each node $\in N$ (this includes places belonging to $start$ and $end$);
- For each edge $(s, t) \in E$, a subgraph connecting the place belonging to $s$ to the place belonging to $t$. This subgraph depends on whether $t$ is $end$.
  If $t$ is $end$, then this subgraph is a silent transition connecting $s$ to $t$:

  If $t$ is not $end$, then this subgraph executes $t_s$ and $t_c$ in sequence, where $t_s$ is optional:

  This last step ensures that start and completion events are accounted for in the alignments, while keeping the start events optional (as these are not always present).

For instance, Figure 3 shows the Petri net as translated from the DFM in Figure 2. Given this translation procedure, we can show that if a sound DFM is translated, a sound workflow net is obtained:

*Proof of Lemma 1.* Consider the translation of Section V-A: By the $start$- and $end$-nodes (Definition 1) and Definition 3, the translated Petri net is a workflow net. As the translation does not introduce transitions with multiple outgoing or incoming edges, the workflow net is 1-bounded (safe). Hence, the translated workflow net is sound. Language equivalence follows by construction of the translation. Hence, the translated Petri net is a language-equivalent sound workflow net. □

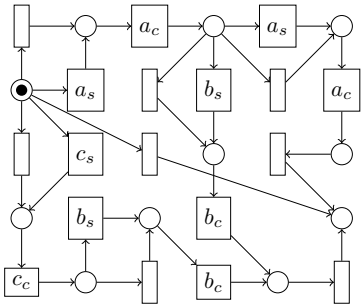



Fig. 3. Example of the DFM of Figure 2 translated to a Petri net.

Once a sound DFM has been translated into a workflow net, any standard Petri net-based conformance checking technique can be applied. For instance, fitness [17], [19], precision [26], [19] and simplicity can be measured, which makes Trace-Based DFM discovery a "full" discovery technique, which could be compared with other process discovery techniques. However, such a qualitative comparison is outside the scope of this paper.

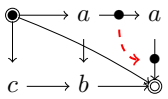
### B. Alignment Visualisation

Using a Petri net and the event log, alignments can be computed. Alignments have been visualised on Petri nets [17] and process trees [20]. For DFMs, one could use the Petri net alignment visualisation, however this would require users to familiarise themselves with a new visualisation of their DFM.

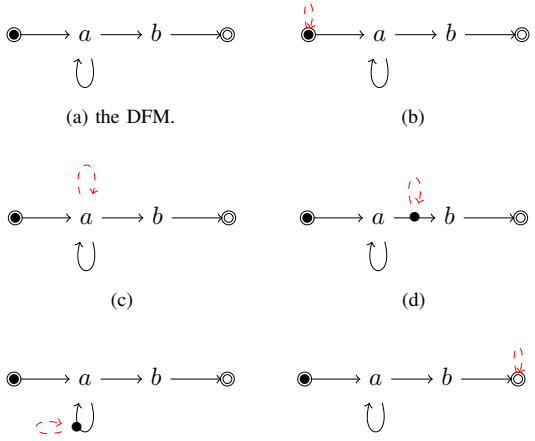
Alignments result in two types of deviations: model moves and log moves. We discuss the visualisation of both types of deviation in more detail. Model moves are visualised with an edge bypassing a node in the model, indicating that an event should have been executed according to the model, but it was not seen in a trace of the event log. For instance, consider the DFM of Figure 2 and the trace $\langle a_s, a_c \rangle$. Then, an optimal alignment is

| log | $a_s$ | $a_c$ | - |
|---|---|---|---|
| model | $a_s$ | $a_c$ | $a_c$ |

, which can be visualised as follows on the DFM:




Log moves indicate that an event appeared in the log while it was not allowed at that point by the model. We identified five positions on which a log move can occur, and we illustrate them using the following DFM as shown in Figure 4a.
(1) At the start of a trace, which we visualise by a loop on the start node. For instance, the trace $\langle l_c, a_s, a_c, b_s, b_c \rangle$ yields an alignment that is visualised in Figure 4b.
(2) During execution of $a$, which we visualise by a loop on $a$. For instance, the trace $\langle a_s, l_c, a_c, b_s, b_c \rangle$ yields an alignment that is visualised in Figure 4c.
(3) In between the executions of $a$ and $b$, which we visualise by a loop on the edge from $a$ to $b$. For instance, the trace $\langle a_s, a_c, l_c, b_s, b_c \rangle$ yields an alignment that is visualised in Figure 4d.
(4) In between two executions of $a$, which we visualise by a loop on the model's loop on $a$. For instance, the trace




(a) the DFM. (b) (c) (d) (e) (f)

Fig. 4. A DFM and log move positions on it.

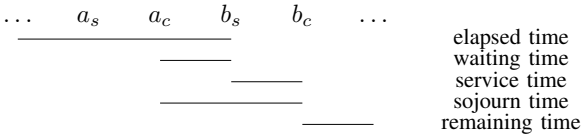
$\langle a_s, a_c, l_c, a_s, a_c, b_s, b_c \rangle$ yields an alignment that is visualised in Figure 4e.

(5) At the end of a trace, which we visualise by a loop on the end node. For instance, the trace $\langle a_s, a_c, b_s, b_c, l_c \rangle$ yields an alignment that is visualised in Figure 4f.

By construction, each model or log move corresponds to one of these types.

## VI. DF-Performance Measures

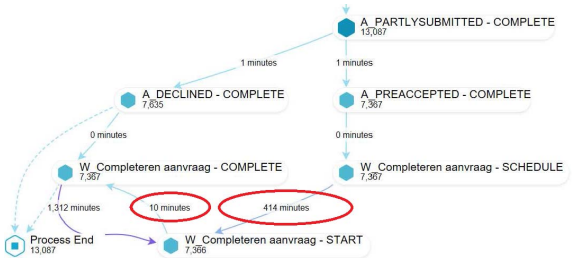
Another perspective that is often used in process mining is the performance perspective. DFMs enable the computation of most commonly used performance indicators, such as service time, waiting time and sojourn time [27]. In addition, the time elapsed and remaining in a case can be of interest:




Synchronisation time [1] is not defined on DFMs, as DFMs cannot represent concurrent activities.

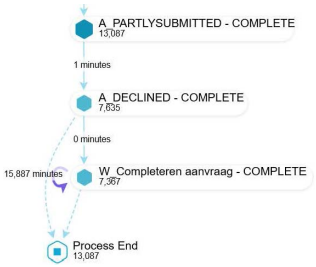
Many process mining tools provide performance measures. However, measures can be unreliable without the use of sound models and alignments. That is, existing approaches, while removing edges from a directly follows-based model, also remove the time that was associated with that edge from the model. As a result of this, the times shown do not add up correctly to overall case durations and times seem to disappear.

For instance, consider the directly follows-based models shown in Figure 5: these models are derived from the same event log using different levels of filtering in CPE. In these models, consider the path taken through `W_Completeren aanvraag-schedule` via `-start` to `-complete`. In the least-filtered model (Figure 5a), this path takes more than 400 minutes. In a bit more filtered model (Figure 5b), this path takes 88 minutes, while in the most-filtered model (Figure 5c), it does not seem to take




(a) 36.6% activities filtering, 31.2% connections filtering.


(b) 27.7% activities filtering, 17.7% connections filtering.




(c) 22% activities filtering, 13.3% connections filtering.

Fig. 5. BPI Challenge 2012 log in CPE 4.4 at different levels of filtering.

any time. In fact, in the most-filtered model, seemingly the majority of cases takes only 1 minute, which could lead to the false conclusion that barely any time is spent in the process. A further consideration found similar issues in the other commercial process mining tools.

By using sound Petri nets and alignments to compute performance measures, such issues can be avoided. For instance, the time between `-schedule` and `-start` would be added to the time between `-schedule` and `-complete` if `-start` would be filtered out [27], [19], [20]. This highlights the importance of reliable performance measures *and conformance checking*, which could be of aid in signalling that the path `A_PARTLYSUBMITTED-complete` via `A_PREACCEPTED-complete` to `W_Completeren aanvraag-complete` has been removed from the model by filtering[4].

*Directly Follows visual Miner:* To convey the concepts of DFM discovery, conformance checking and performance measuring, these concepts have been implemented in a new

[4]CPE possesses conformance checking capabilities, but not on its DFM.

tool, the *Directly Follows visual Miner* (DFvM). Given an event log, DFvM first discovers a DFM, translates it to a Petri net, aligns this net, visualises deviations and animates the event log on the DFM, all without user interaction. Users can change parameters, filter the log, change the visualisation, etc., after which DFvM will recompute the necessary steps automatically, enabling a truly exploratory approach to process mining. Figure 7 shows DFvM, which is available as a plug-in of the ProM framework [28] and shares its code base with the Inductive visual Miner [20]. Additionally, the plug-in *Visualise deviations on Directly Follows Model* allows for easy comparison of a DFM model and an event log, that is, this plug-in entails DFvM without the discovery parts but with a given model.

## VII. Case Study at the Queensland Government

One of the motivators for this approach was a case study that was performed with a major government organisation in Australia. The organisation relies on an IT service management system to automatically capture activities performed by operational teams. The case study focused on financial and human resources processes support by this system. Documentation for some of these processes existed and but for others only outdated or no models existed. During each of the calendar years, 2017 and 2018, the organisation instigated several initiatives to improve the quality of their service performance.

The organisation had three objectives: i) to have complete and up-to-date documentation of their processes, ii) to check whether processes are executed correctly and iii) to identify performance bottlenecks in their processes. To achieve these objectives we extracted an event log for one month of operations covering four areas of operations with up to 53 distinct workflows each. The log contained 142896 active cases of which 80883 were completed and used in the analysis.

To achieve the first objective, the new approach was used to mine direct-follows process models from the event log. As all of the processes are sequential and do not contain concurrent tasks, direct-follows models appropriately document the processes for which documentation was missing or outdated.

To achieve the second objective, DFMs were automatically constructed from Excel tables specifying the intended process as lists of direct-follows-relations between process activities. As existing approaches could not do conformance checking for DFMs, we used the new approach to check and annotate the constructed models with conformance information, as shown in Figure 6. This example model shows several unexpected transitions (red dashed lines) between activities which were missing in the documentation and proved that the existing documentation is not up-to-date.

The third objective was to identify performance bottlenecks in the analysed processes. Existing approaches produced performance data that was too complex without filtering and became inconsistent on filtering. Instead, we used our mined direct-follows models for the analysis. Figure 7 shows a model with performance annotations (sojourn times). Activities are

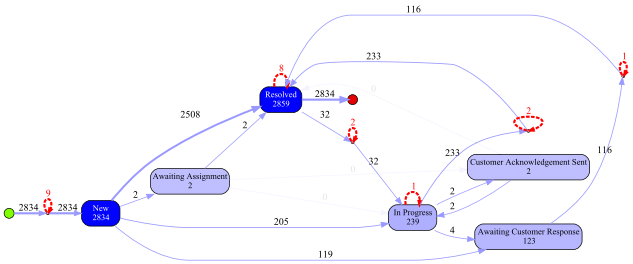



Fig. 6. DFM annotated with conformance and frequency information; labels are not intended to be readable.

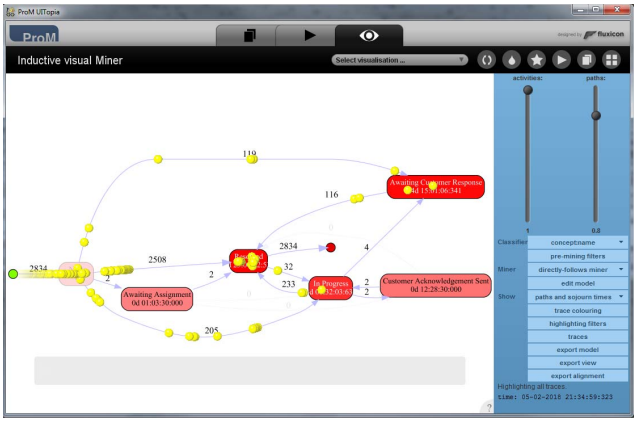



Fig. 7. Screenshot of the Directly Follows visual Miner (DFvM) in ProM, where animated tokens show cases flowing through the process. The DFM is annotated with performance information (sojourn times).

drawn in red to indicate durations. For example, cases involving interactions with external customers take significantly longer.

The stakeholders were also interested in comparing cases across various groupings such as request types, so annotated models were generated for each workflow and each of these groupings, resulting in a total of 4905 models. Data from these models was aggregated and some high-impact workflows were identified and analysed in more detail, and all the models were handed over to the organisation for internal documentation.

At the time of writing the organisation's business improvement team is incorporating the analysis results in their review of business process effectiveness and efficiency. When results were presented, directly-follows models highlighted deviations from documented process pathways. Further deviations became apparent to business improvement team members during the visual replays of some of the business process models using DFvM in ProM. Team members discussed likely explanations for such deviations including incorrect uses of the IT service management system, data quality issues and potential training issues. The business improvement team will use the conformance checking results to identify and rectify potential data processing errors and associated data quality issues. The results also highlighted high-level processes with long-running sub-processes. The business improvement team will review each of these process and its sub-processes for underlying causes. The business improvement team reviewed a sample of the new process documentation generated by process

mining. The review highlighted that some process activities require additional effort to increase documentation.

## VIII. Conclusion

While many process mining tools exist, many commercial tools lack conformance checking (quality of models cannot be verified), their performance measures might be unreliable, and academic tools lack the intuitiveness and simplicity of DFMs.

Our approach bridges this gap by showing how DFMs can be discovered from event logs, while guaranteeing soundness and a minimum fitness level chosen by the user. We introduced a DFM-to-Petri-net translation, which is used to align the model with a log. The alignment is then used to measure the quality of a discovered model, identify and visualise deviations, and provide reliable performance measures. Furthermore, we highlighted issues in performance measures of existing commercial process mining tools and showed that alignment-based measures do not suffer from the same issues.

We implemented discovery, conformance checking and performance measures based on DFMs in a process mining tool that enables exploring event logs by repeated discovery, conformance checking and filtering.

This tool was applied in a case study in the Queensland Government. In this study, we discovered DFMs from event logs, compared them to existing manually made DFMs and analysed the performance of processes; the outcomes of this study were used to document, review, improve and automate processes. Overall, we showed that conformance checking and reliable performance measures can be combined with directly follows-based models and that such models can provide guarantees such as soundness or a minimum fitness level. This tool demonstrates that conformance checking and transparent performance measures can be offered in typical process mining tools based on directly follows semantics. We hope that commercial parties will adopt these ideas to enable their customers to assess the quality of discovered models and to better understand performance measures.

In future work, it would be interesting to evaluate how our discovery technique compares to existing discovery techniques that support advanced constructs. Furthermore, discovered DFMs might be used as input to these techniques [29]. Furthermore, advanced infrequent behaviour filtering techniques might improve results or provide different insights [25]. Finally, our conformance checking concepts might be used to perform conformance checking on hybrid models [2].

*Acknowledgement*: We gratefully acknowledge funding and support from the Queensland Government during the conduct of this research.

## References

[1] W. M. P. van der Aalst, *Process Mining - Data Science in Action, Second Edition*. Springer, 2016.

[2] W. M. P. van der Aalst, R. De Masellis, C. D. Francescomarino, and C. Ghidini, "Learning hybrid process models from events - process discovery without faking confidence," in *BPM*, 2017, pp. 59–76.

[3] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering block-structured process models from event logs containing infrequent behaviour," in *BPM workshops*, 2013, pp. 66–78.

[4] A. Augusto, R. Conforti, M. Dumas, and M. L. Rosa, "Split miner: Discovering accurate and simple business process models from event logs," in *IEEE ICDM*, 2017, pp. 1–10.

[5] W. Reisig, *A primer in Petri net design*, ser. SCI. Springer, 1992.

[6] O. M. G. (OMG), "Business process model and notation (BPMN) version 2.0," Object Management Group (OMG), Tech. Rep., jan 2011.

[7] M. De Leoni, F. M. Maggi, and W. M. P. van der Aalst, "Aligning event logs and declarative process models for conformance checking," in *BPM*. Springer, 2012, pp. 82–97.

[8] A. Weijters, W. van der Aalst, and A. de Medeiros, "Process mining with the heuristics miner-algorithm," Eindhoven University of Technology, BETA Working Paper series 166, 2006.

[9] C. W. Günther and W. M. P. van der Aalst, "Fuzzy mining - adaptive process simplification based on multi-perspective metrics," in *BPM*, 2007, pp. 328–343.

[10] R. Conforti, M. Dumas, L. García-Bañuelos, and M. L. Rosa, "BPMN miner: Automated discovery of BPMN process models with hierarchical structure," *Inf. Syst.*, vol. 56, pp. 284–303, 2016.

[11] S. K. L. M. vanden Broucke and J. D. Weerdt, "Fodina: A robust and flexible heuristic process discovery technique," *Decision Support Systems*, vol. 100, pp. 109–118, 2017.

[12] J. C. A. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst, "A genetic algorithm for discovering process trees," in *IEEE CEC*, 2012, pp. 1–8.

[13] V. Liesaputra, S. Yongchareon, and S. Chaisiri, "Efficient process model discovery using maximal pattern mining," in *BPM*, 2015, pp. 441–456.

[14] A. Ehrenfeucht and G. Rozenberg, "Partial (set) 2-structures. part I: basic notions and the representation problem," *Acta Inf.*, vol. 27, no. 4, pp. 315–342, 1990.

[15] C. W. Günther and A. Rozinat, "Disco: Discover your processes," in *BPM demos*, 2012, pp. 40–44.

[16] V. Leno, A. Armas-Cervantes, M. Dumas, M. L. Rosa, and F. M. Maggi, "Discovering process maps from event streams," in *ICSSP 2018*, 2018, pp. 86–95.

[17] W. M. P. van der Aalst, A. Adriansyah, and B. F. van Dongen, "Replaying history on process models for conformance checking and performance analysis," *WIRDMKD*, vol. 2, no. 2, pp. 182–192, 2012.

[18] M. T. Wynn, E. Poppe, J. Xu, A. H. M. ter Hofstede, R. Brown, A. Pini, and W. M. P. van der Aalst, "Processprofiler3d: A visualisation framework for log-based process performance comparison," *Decision Support Systems*, vol. 100, pp. 93–108, 2017.

[19] A. Adriansyah, "Aligning Observed and Modeled Behavior," Ph.D. dissertation, Eindhoven University of Technology, 2014.

[20] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Process and deviation exploration with Inductive visual Miner," in *BPM Demo Sessions 2014*, 2014, p. 46.

[21] A. Partington, M. T. Wynn, S. Suriadi, C. Ouyang, and J. Karnon, "Process mining for clinical processes: A comparative analysis of four australian hospitals," *ACM TMIS*, vol. 5, no. 4, pp. 19:1–19:18, 2015.

[22] W. M. P. van der Aalst, S. Guo, and P. Gorissen, "Comparative process mining in education: An approach based on process cubes," in *DDPDA*. Springer, 2013, pp. 110–134.

[23] R. P. J. C. Bose, A. Gupta, D. Chander, A. Ramanath, and K. Dasgupta, "Opportunities for process improvement: A cross-clientele analysis of event data using process mining," in *ICSOC*, 2015, pp. 444–460.

[24] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Using life cycle information in process discovery," in *BPM Workshops*, 2015, pp. 204–217.

[25] M. F. Sani, S. J. van Zelst, and W. M. P. van der Aalst, "Improving process discovery results by filtering outliers using conditional behavioural probabilities," in *BPM workshops*, 2017, pp. 216–229.

[26] A. Adriansyah, J. Munoz-Gama, J. Carmona, B. F. van Dongen, and W. M. P. van der Aalst, "Measuring precision of modeled behavior," *Inf. Syst. E-Business Management*, vol. 13, no. 1, pp. 37–67, 2015.

[27] M. Leemans, W. M. P. van der Aalst, and M. G. J. van den Brand, "Hierarchical performance analysis for process mining," in *ICSSP*, 2018, pp. 96–105.

[28] B. F. van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and W. M. P. van der Aalst, "The ProM framework: A new era in process mining tool support," in *Petri Nets*, 2005, pp. 444–454.

[29] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Scalable process discovery and conformance checking," *SoSyM*, vol. 17, no. 2, pp. 599–631, 2018.

# Inductive Context-aware Process Discovery

Roee Shraga*, Avigdor Gal*, Dafna Schumacher*, Arik Senderovich† and Matthias Weidlich‡
*Technion – Israel Institute of Technology, †University of Toronto, ‡Humboldt-Universität zu Berlin
{shraga89@campus., avigal@, dafna.s@campus.}technion.ac.il, sariks@mie.utoronto.ca, matthias.weidlich@hu-berlin.de

***Abstract*—Discovery plays a key role in data-driven analysis of business processes. The vast majority of contemporary discovery algorithms aims at the identification of control-flow constructs. The increase in data richness, however, enables discovery that incorporates the context of process execution beyond the control-flow perspective. A "control-flow first" approach, where context data serves for refinement and annotation, is limited and fails to detect fundamental changes in the control-flow that depend on context data. In this work, we thus propose a novel approach for combining the control-flow and data perspectives under a single roof by extending inductive process discovery. Our approach provides criteria under which context data, handled through unsupervised learning, take priority over control-flow in guiding process discovery. The resulting model is a process tree, in which some operators carry data semantics instead of control-flow semantics. We evaluate the approach using synthetic and real-world datasets and show that the resulting models are superior to state-of-the-art discovery methods in terms of measures that are based on multi-perspective alignments.**

## I. Introduction

Process discovery plays a key role in descriptive, predictive, and prescriptive data-driven process analyses [1]. From depicting the underlying processes [2], [3], through providing a basis for predictive monitoring [4] and conformance checking [5], to evidence-based resource scheduling [6], discovered models serve as the workhorse of process-aware data analytics.

In recent years, the focus of process discovery has been on extending the control-flow perspective to additional perspectives that represent the context of process execution, *e.g.*, related to time and costs. Most multi-perspective discovery methods proceed in two phases [1]: (1) eliciting the control-flow, and (2) enriching the discovered models with context information. This separation may result in accuracy loss when data-based quality measures are concerned. To demonstrate the challenge, consider a business process from the life of a University faculty Prof. X, who works 7:00-20:00 and his efficient assistant, Wolverine (W), who works 9:00-12:00 (an event log snippet is presented in Table I). Prof. X, being a research traditionalist, scribbles his research notes on paper (activity A) and sends it to his assistant (activity B), who types the notes using a computer (activity C). When Wolverine heads home, Prof. X must perform voice-to-text typing on his own (activity D).

Using a control-flow-oriented discovery algorithm, *e.g.*, the inductive miner [3], for this scenario yields the process tree in Figure 1a. Clearly, the algorithm fails to uncover the dependencies between control-flow and execution context, such as times and resources, being guided solely by the activity labels of the events in the log. The alternative we offer in this work is a context-aware approach, which yields a process tree

**TABLE I:** Example event log.

| | Case | Activity | Resource | Time |
|---|---|---|---|---|
| $e_1$ | 1 | A | X | 11:00 |
| $e_2$ | 1 | B | X | 11:00 |
| $e_3$ | 1 | C | W | 11:10 |
| $e_4$ | 2 | A | X | 18:00 |
| $e_5$ | 2 | D | X | 18:30 |
| $e_6$ | 3 | A | X | 10:00 |
| $e_7$ | 3 | B | X | 10:00 |
| $e_8$ | 3 | C | W | 10:10 |
| $e_9$ | 4 | A | X | 9:00 |
| $e_{10}$ | 4 | B | X | 9:00 |
| $e_{11}$ | 4 | C | W | 9:05 |

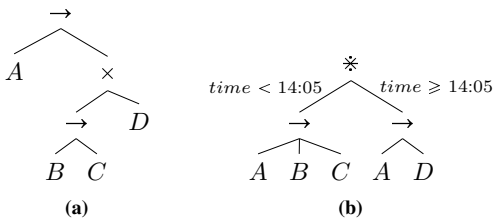



**Fig. 1:** Process trees constructed from the example event log in Table I with (a) traditional inductive discovery, and (b) context-aware inductive discovery.

as shown in Figure 1b. Our algorithm first splits a log based on context information, discovering distinct processes at different times of the day (colored yellow and gray in Table I). This is reflected in Figure 1b, where a node in the tree uses a dedicated operator that carries data semantics: It describes processes that are either executed before or after 14:05. Processing the sublogs further, potential data-based splits (*e.g.*, based on the resource information, colored dark and light gray in Table I) turn out to be insufficiently "significant" to take precedence over the control-flow perspective. Hence, control-flow dependencies govern the construction of the subtrees.

The above observation is also important for performance-oriented process discovery. Based on the tree in Figure 1a and the timestamps recorded in Table I, we may arrive at the conclusion that the average duration of the process is 13.75min. However, based on Figure 1b, significant differences are observed for executions before 14:05 (average duration of 8.3min) and later (average duration of 30min). Hence, the ability to differentiate the two scenarios increases the accuracy of the discovered model, when used for performance analysis.

To improve the accuracy of context-aware process discovery, we propose an inductive discovery technique that elicits both the control-flow and the data perspective, simultaneously. To this end, we combine the divide-and-conquer approach of the

978-1-7281-0919-0/19/$31.00 ©2019 IEEE
DOI 10.1109/ICPM.2019.00016

inductive miner [3] with ideas from unsupervised learning. For discovering context, we define *context-aware process trees*, a representational bias that captures both control-flow and data-driven decisions. We use multi-perspective alignments [7] to measure the quality of discovered models (via precision, fitness, and generalization) to demonstrate, empirically, that our model yields higher quality than a "control-flow first" approach.

Our main contributions can be summarized as follows.

1) Defining context-aware process trees (Section III).
2) Introducing an inductive context-aware discovery algorithm (Section IV).
3) Providing an empirical evaluation based on synthetic and real-world datasets to demonstrate the value of our approach (Section V)

In addition, we discuss the relation of our work to state-of-the-art literature in Section VI and conclude in Section VII.

## II. Background

We first define our data model (event logs) and the adopted process modelling formalism (process trees), and briefly outline the main ideas of the inductive miner [3] to discover process trees from event data. Subsequently, we discuss log clustering and the *silhouette* evaluation measure.

### A. Event Logs

We adopt a common model of event logs, grounded in sequences of events that represent activity executions as part of a single process instance (aka case) [1]. Let $\mathcal{E}$ and $\mathcal{A}$ be the universes of *events* and *activities*, respectively. Each event $e \in \mathcal{E}$ represents the execution of an activity $e.a \in \mathcal{A}$ and carries information on the context in which the activity was executed. This context is defined by a set of attributes $\mathcal{X} = \{x_1, \ldots, x_p\}$, with $dom(x_i)$ as the domain of attribute $x_i$ ($1 \leqslant i \leqslant p$). We write $e.x$ for the value of attribute $x$ of event $e$. An event is assumed to have at least a timestamp ($e.time$), but may also include further attributes, such as the resource that executed the activity ($e.resource$).

A *trace* $\sigma = \langle e_1, \ldots, e_n \rangle \in \mathcal{E}^*$ is a finite sequence of events corresponding to a single process instance, with $|\sigma|$ denoting its length $n$. An *event log* is a set of traces $L = \{\sigma_1, \ldots, \sigma_m\}$, $\sigma_i \in \mathcal{E}^*$, $1 \leqslant i \leqslant m$, where no event occurs in more than one trace. We write $\mathcal{L}$ for the universe of event logs.

A trace $\sigma$ induces a sequence of activities, $\sigma^a = \langle e_1.a, \ldots, e_n.a \rangle$, so that an event log $L$ may also be regarded as a multi-set of activity sequences, $L^a : \mathcal{A}^* \to \mathbb{N}$, with $L^a(x) = |\{\sigma \in L \mid \sigma^a = x\}|$. The set representation of $L^a$, denoted as $\bar{L}^a$, is the set of unique activity sequences in $L^a$.

For example, using the event log from Table I, $L = \{\langle e_1, e_2, e_3 \rangle, \langle e_4, e_5 \rangle, \langle e_6, e_7, e_8 \rangle, \langle e_9, e_{10}, e_{11} \rangle\}$ and $L^a = [\langle A, B, C \rangle^3, \langle A, D \rangle^1]$, meaning that the log comprises three traces that induce an activity sequence $\langle A, B, C \rangle$, and one trace that induces $\langle A, D \rangle$. Then, the set representation contains two activity sequences, $\bar{L}^a = \{\langle A, B, C \rangle, \langle A, D \rangle\}$.

An operator set $\bigoplus$ for activity sequences contains a set of functions such that $\oplus \in \bigoplus$ maps several sequences to a single sequence, $\oplus : \mathcal{A}^* \times \ldots \times \mathcal{A}^* \to \mathcal{A}^*$. For example, the concatenation operator $\oplus_{\rightarrow} \in \bigoplus$ concatenates the given sequences. Given two activity sequences, $\sigma_1^a = \langle A, B \rangle$ and $\sigma_2^a = \langle C, D \rangle$, $\sigma_1^a \oplus_{\rightarrow} \sigma_2^a = \langle A, B, C, D \rangle$.

$\pi_{\oplus}(L)$ is a split of an event log $L$ into sublogs with respect to a set of operators $\bigoplus$, defined as follows:

$$\pi_{\oplus}(L) = \left\{ L_1, \ldots, L_M \in \mathcal{L} \mid \bar{L}^a = \bigoplus_{i=1}^{M} \bar{L}^a{}_i \right\} \subseteq 2^{\mathcal{L}}. \quad (1)$$

According to Eq. (1), the set representation of activity sequences $\bar{L}^a$ of the log can be reconstructed by applying a sequence of (possibly different) operators from $\bigoplus$ on the set representations of activity sequences of $L_1^a, \ldots, L_M^a$.

### B. Process Trees and their Discovery

A *process tree* is a rooted tree that represents a process, where each leaf node is an activity and all non-leaf nodes are control-flow operators [3]. Common control-flow operators include $\rightarrow$ (sequence), $\times$ (exclusive choice), $\wedge$ (concurrency), and $\circlearrowleft$ (loop). Process trees are defined recursively. Let $\Phi = \{\rightarrow, \times, \wedge, \circlearrowleft\}$ be a set of *operators* and $\tau \notin \mathcal{A}$ be the *silent activity*. Then, $a \in \mathcal{A} \cup \{\tau\}$ is a process tree; and $\phi(T_1, \ldots, T_n)$, $n > 0$, with $T_1, \ldots, T_n$ being process trees and $\phi \in \Phi$ being an operator, is a process tree ($n > 1$ if $\phi = \circlearrowleft$). The universe of process trees is denoted by $\mathcal{M}_T$.

For a process tree $T$, the execution semantics is defined by a set of sequences of activities. This set, the *language* of $T$, is defined recursively and captured by a function $\nu : \mathcal{M}_T \to 2^{\mathcal{A}^*}$ that assigns sets of activity sequences to process trees. Trivially, $\nu(a) = \{\langle a \rangle\}$ for $a \in \mathcal{A}$ and $\nu(\tau) = \{\langle\rangle\}$. For an operator $\phi \in \Phi$, semantics is defined by a language join function $\phi_l : 2^{\mathcal{A}^*} \times \ldots \times 2^{\mathcal{A}^*} \to 2^{\mathcal{A}^*}$. It defines the execution semantics of a process tree $\phi(T_1, \ldots, T_n)$ as $\nu(\phi(T_1, \ldots, T_n)) = \phi_l(\nu(T_1), \ldots, \nu(T_n))$. As an example, the language induced by the exclusive choice operator $\times_l(\Sigma_1, \ldots, \Sigma_n)$ is given by the union of the languages of its children $\bigcup_{1 \leqslant i \leqslant n} \Sigma_i$, see [3] for further details.

An example process tree is given in Figure 1a. Its language is given as the set of activity sequences $\{\langle A, B, C \rangle, \langle A, D \rangle\}$.

A prominent approach for discovering process trees from event logs is the *inductive miner* (IM) [3], [8]. The IM algorithm recursively applies a *select* function $\eta : \mathcal{L} \to \Phi \times 2^{\mathcal{L}}$, which, given an event log, selects a tree operator $\phi \in \Phi$ and splits the log into $\pi_{\{\oplus_\phi\}}(L)$. The IM algorithm starts by applying the select function $\eta$ on a given event log $L$. A tree operator $\oplus_\phi$ is returned, along with the logs, $L_1^1, \ldots, L_{v_1}^1$ that result from the split $\pi_{\{\oplus_\phi\}}(L)$. Subsequently, the select function is applied to every resulting event log $L_i^1$ recursively, until a base case of a log containing single-event traces is reached.

### C. Clustering

Clustering is an unsupervised learning technique aimed at grouping objects that are more similar to each other than other objects. In this work, we adopt the *K-means* algorithm [9], a well established clustering method [10]. While an event log is split into sets of traces, we first cluster individual events and,

based thereon, derive the clustering of traces. This way, we achieve clustering solely based on context information.

For an event log $L$, let $E_t = \{e_1, \ldots, e_n\}$ be the set of events of a trace $t = \langle e_1, \ldots, e_n \rangle \in L$, and $E_L = \bigcup_{t \in L} E_t$ be the set of events of all traces. With $\mathcal{X} = \{x_1, \ldots, x_p\}$ as the set of attributes that model context information, each event $e$ can be seen as a $p$-dimensional feature vector $(e.x_1, \ldots, e.x_p)$.

Applying $K$-means clustering to a set of events, we need to be able to quantify the distance between two events $e_i$ and $e_j$. Since the respective feature vectors contain numeric and categorical attributes, a mixed-type measure is needed. Following [11], we separate the categorical attributes $\mathcal{X}_C \subseteq \mathcal{X}$ (*e.g.,* resource) and the numeric attributes (*e.g.,* time). The distance for features of numeric attributes is computed by their Euclidean distance, whereas for categorical attributes, we assess the equivalence of the respective values:

$$d_e(e.x, e'.x) = \begin{cases} 0, & \text{if } e.x = e'.x, \\ 1, & \text{if } e.x \neq e'.x. \end{cases}$$

The mixed type measure for two events is defined as follows:

$$d(e, e') = \omega \sqrt{\sum_{x \in \mathcal{X}_N} (e.x - e'.x)^2} + (1 - \omega) \sum_{x \in \mathcal{X}_C} d_e(e.x, e'.x).$$

We set $\omega$ to be the proportion of numeric features out of all features, *i.e.,* $\omega = |\mathcal{X}_N|/|\mathcal{X}|$. Given a set of events $E \subseteq E_L$, a centroid may be calculated. Again, numeric and categorical attributes need to be distinguished. For numeric attributes $\mathcal{X}_N = \{x_1, \ldots, x_m\}$, the numeric centroid is defined as

$$\mu_N(E) = \left( \frac{\sum_{e \in E} e.x_1}{|E|}, \cdots, \frac{\sum_{e \in E} e.x_m}{|E|} \right).$$

For categorical attributes $\mathcal{X}_C = \{x_1, \ldots, x_m\}$, the categorical centroid is obtained by majority vote:

$$\mu_C(E) = \left( \underset{x \in \bigcup_{e \in E} \{e.x_1\}}{\arg\max} (x), \cdots, \underset{x \in \bigcup_{e \in E} \{e.x_m\}}{\arg\max} (x) \right).$$

By concatenating the numeric centroid, $\mu_N$, and the categorical centroid, $\mu_C$, we obtain the centroid of a set of events $E \subseteq E_L$, denoted by $\mu(E)$. Although $\mu(E)$ does not represent an actual event, it is a $p$-dimensional feature vector. Hence, for an event $e$, we can compute an event-to-centroid distance $d(e, \mu(E))$.

Based thereon, $K$-means clustering first assigns each event randomly to one of $K$ distinct subsets and computes a centroid $\mu(E_j)$ for each subset $E_j$. It then assigns each event $e$ to the subset $E_j$, for which the distance $d(e, \mu(E_j))$ is minimal. This is done repeatedly, until none of the subsets changes.

Using the obtained event clusters $\{E_1, \ldots, E_K\}$, we split the event log into sublogs, *i.e.*, sets of traces. For each trace $t = \langle e_1, \ldots, e_n \rangle$, an event cluster $E_j$ induces a projected trace $t_{|E_j} = \langle e'_1, \ldots, e'_m \rangle$ that contains all events of $E_j$ in $t$, $E_{t_{|E_j}} = E_t \cap E_j$, preserving their respective order. Lastly, $\pi_{K-means}(L) = \{L_1, \ldots, L_K\}$ splits $L$ into sublogs, such that $L_j = \bigcup_{t \in L} \{t_{|E_j}\}$.

Figure 2 shows the clustering result for the running example in Table I for $K = 2$. Clustering split the event log into two sets

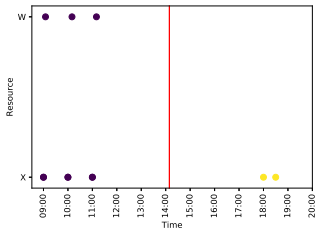



**Fig. 2:** Example of $K$-means clustering, $K = 2$, for the events of the log in Table I for the time and resource attributes.

of events (gray and yellow dots), with a red line illustrating the boundary between both clusters in terms of the time attribute.

To evaluate the quality of the result of clustering, in the absence of a reference split that can be used for comparison, we shall use the *silhouette* coefficient [12]. It quantifies the separation between clusters, thereby providing us with a data-oriented quality measure of a split of the events of a log. Given a set of event clusters $\{E_1, \ldots, E_K\}$, for an event $e \in E_j$ $(1 \leqslant j \leqslant K)$, the silhouette coefficient $\psi_s(e)$, is defined as:

$$in(e) = \frac{\sum_{e' \in E_j \setminus \{e\}} d(e, e')}{|E_j|} \quad out(e) = \min_{1 \leqslant i \leqslant K, i \neq j} \frac{\sum_{e' \in E_i} d(e, e')}{|E_i|}$$

$$\psi_s(e) = \frac{out(e) - in(e)}{\max(in(e), out(e))}$$

The silhouette coefficient of a set of event clusters $\{E_1, \ldots, E_K\}$ is calculated by averaging over all events of all clusters:

$$\psi_s(\{E_1, \ldots, E_K\}) = \frac{1}{K} \sum_{1 \leqslant j \leqslant K} \frac{\sum_{e \in E_j} \psi_s(e)}{|E_j|}$$

Above, we detailed how a set of event clusters $\{E_1, \ldots, E_K\}$ is used to split an event log $L$ into sublogs, $\pi_{K-means}(L) = \{L_1, \ldots, L_K\}$. As such, we lift the silhouette coefficient and use it as an evaluation of a set of sublogs, $\psi_s(\pi_{K-means}(L)) = \psi_s(\{E_1, \ldots, E_K\})$.

For the above example in Figure 2, the silhouette coefficient for the obtained clusters of events is 0.77.

## III. Context-Aware Process Trees

We introduce *context-aware process trees* (CaT) as our representational bias, *i.e.*, as the target formalism for our discovery algorithm. CaTs extend process trees (Section II-B) with an operator that carries data semantics, instead of control-flow semantics. Therefore, a CaT may feature a constraint operator that restricts the contexts in which subtrees, and thus activities, may be executed. For instance, in the tree in Figure 1b, activities $A$, $B$, and $C$ may be executed in a sequence, but only in a specific temporal context (before 14:05). The sequence of activities $A$ and $D$, in turn, can only be executed after the respective point in time. Put differently, the constraint operator restricts the set of possible values of

an attribute (*e.g.*, *resource* or *time*) that can be assigned to an activity execution in a subtree.

A CaT is not merely a process tree with branching conditions, as they may be derived by control-flow discovery followed by decision mining [13]. A traditional choice operator that is extended with branching conditions grounds the choice between different control-flow continuations at one particular state, but does not constrain the context of subsequent activities executions. It does not restrict the sets of values assigned to attributes of subsequent activity executions, whereas in a CaT, such restrictions may be enforced by upper levels of the tree.

To formalize the model of context-aware process trees, recall (Section II-A) that $\mathcal{X} = \{x_1, \ldots, x_p\}$ refers to a set of attributes that describe the context of activity execution, with $dom(x_i)$ being the domain of attribute $x_i$, $1 \leqslant i \leqslant p$. Assuming some total ordering of attributes, $(x_1, \ldots, x_p)$, a *constraint* over $\mathcal{X}$ is modeled as $\rho \subseteq dom(x_1) \times \ldots \times dom(x_p)$ restricting the value combinations that may be assumed by the attributes. While this notion defines the context explicitly by enumerating possible value combinations, we will use hereinafter predicates as a declarative characterization of constraints. For example, for attributes $\{resource, time\}$, we write $time < 14{:}05$ to denote the constraint that does not restrict the domain of attribute $resource$, while timestamps must come from $\{0{:}00, 0{:}01, \ldots, 14{:}04\}$. By $\Delta(\mathcal{X})$, we denote the universe of all possible constraints over $\mathcal{X}$.

A *constraint operator*, denoted as $\circledast$, is an $m$-ary tree operator that is parametrized by a sequence of constraints $\langle \rho_1, \ldots, \rho_m \rangle$, where the number of constraints corresponds to the number of children of the operator. Then, a *context-aware process tree* is a process tree built from $\Phi = \{\rightarrow, \times, \wedge, \circlearrowleft, \circledast\}$ with $\circledast$ being parametrized with any constraint from $\Delta(\mathcal{X})$. We denote the universe of CaTs by $\mathcal{M}_C$.

Semantics of context-aware process trees are defined by a set of traces, instead of a set of activity sequences, as discussed for process trees (Section II-B). This is due to the fact that the tree now restricts the context in which activities may be executed. Similarly to traditional process trees, however, semantics are defined recursively by a function $\nu_C : \mathcal{M}_C \rightarrow 2^{\mathcal{E}^*}$ that maps CaTs to sets of traces. Here, the trivial cases are defined as $\nu_C(a) = \{\langle e \rangle \mid e.a = a \ \wedge \ \forall\, x \in \mathcal{X} : e.x \in dom(x)\}$ for $a \in \mathcal{A}$ and $\nu_C(\tau) = \{\langle\rangle\}$.

Moreover, for operators $\{\rightarrow, \times, \wedge, \circlearrowleft\}$, the language join functions defined for process trees are directly lifted to CaTs by joining traces instead of joining sets of activity sequences. Semantics of the constraint operator are also captured by a dedicated language join function, $\nu_C(\circledast(T_1, \ldots, T_m)) = \circledast_l(\nu_C(T_1), \ldots, \nu_C(T_m))$. The latter defines different execution contexts, as follows. With $\langle \rho_1, \ldots, \rho_m \rangle$ as the parameter of a constraint operator $\circledast$, the language join function considers all traces per child that satisfy the respective constraint:

$$\circledast_l(\Sigma_1, \ldots, \Sigma_m) = \bigcup_{1 \leqslant i \leqslant m} \{\langle e_1, \ldots, e_n \rangle \in \Sigma_i \mid \forall\, 1 \leqslant j \leqslant n : (e_j.x_1, \ldots, e_j.x_p) \in \rho_i\}$$

**Algorithm 1** Context-aware Process Discovery (CaDi)

```
1:  procedure CaDi(event log L, quality threshold δ)
2:      Create a CaT T
3:      // Data-based log split (Section II-C)
4:      π ← π_{2-means}(L)
5:      // Assess quality of data-based log split
6:      if ψ_s(π_{2-means}(L)) > δ then
7:          Create ⊛ with ⟨ρ_l, ρ_r⟩ based on Eq.(2)
8:          Add ⊛ to T
9:      else
10:         // Control-flow-based log split (Section II-B)
11:         π ← π_⊕(L)
12:         Create control-flow operator φ and add φ to T
13:     end if
14:     // For each sublog in the selected split
15:     for L' ∈ π do
16:         // Check if sublog is trivial
17:         if L' = ∅ ∨ ∀ e, e' ∈ E_{L'} : e.a = e'.a then
18:             Add τ or activity a, respectively, to T
19:         else
20:             // Recurse for the sublog
21:             Compute CaDi(L', δ) and add it to T
22:         end if
23:     end for
24:     return T
25: end procedure
```

Consider the process tree from Figure 1b. Let $\Sigma_l$ and $\Sigma_r$ denote the languages of the left and right subtrees of the constraint operator. Then, $\Sigma_l$ includes all sequences of events that represent an activity sequence $\langle A, B, C \rangle$, whereas $\Sigma_r$ contains all event sequences for $\langle A, D \rangle$. As such, a sequence of events $\langle e_1, e_5 \rangle$ from Table I (not corresponding to a trace in the log!) would be an element of the language $\Sigma_r$. Due to the time assigned to $e_1$, however, the sequence is discarded when joining $L_l$ and $L_r$ and, thus, not part of $\circledast_l(\Sigma_l, \Sigma_r)$.

## IV. Inductive Context-aware Process Discovery

We now introduce our algorithm for inductive context-aware discovery (dubbed CaDi). It combines inductive control-flow discovery and clustering of events to construct a CaT, a context-aware process tree as introduced in Section III.

Our idea is to introduce data-based decisions into inductive mining. Specifically, at each iteration of inductive mining, we first consider a data-based split of an event log through clustering. If this split turns out to be of high quality, *i.e.*, it exceeds a threshold $\delta$, a constraint operator is derived based on the sublogs and added to the CaT. If the data-based split is of low quality, the algorithm proceeds with traditional control-flow-based log split and adds a control-flow operator to the CaT. As such, the parameter $\delta$ balances the two process perspectives, with higher values increasing the importance of control-flow information, whereas lower values prioritize the context data.

A data-based split is obtained by $K$-means clustering, as introduced in Section II-C. Hence, choosing $K$ enables us to

control the number of the sublogs and, therefore, the arity of the constraint operator $\divideontimes$ to add to the CaT. In this work, however, we set $K = 2$, thereby distinguishing two execution contexts with a constraint operator. Note though, that constraint operators may be nested, so that a discovered CaT may capture more than two contexts.

The two clusters, $E_l$ and $E_r$, obtained with the 2-means algorithm induce two centroids, $\mu_l$ and $\mu_r$, respectively. Based on these centroids, we construct the two constraints of a constraint operator, $\rho_l$ and $\rho_r$, as follows. With $\mathcal{X}_C \subseteq \mathcal{X}$ and $\mathcal{X}_N \subseteq \mathcal{X}$ as the categorical attributes and numeric attributes, respectively, that describe execution contexts, we define the constraints $\rho_l$ and $\rho_r$ as follows:[1]

$$\begin{aligned} \rho_l = & \bigtimes_{x \in \mathcal{X}_N} \{z \in dom(x) \mid z < \frac{\mu_l + \mu_r}{2}\} \times \\ & \bigtimes_{x \in \mathcal{X}_C} \{z \in dom(x) \mid z = \underset{y \in \bigcup_{e \in E_l} \{e.x\}}{\arg\max} (y)\} \\ \rho_r = & \bigtimes_{x \in \mathcal{X}} dom(x) \setminus \rho_l. \end{aligned} \tag{2}$$

Using the above constructions, we introduce inductive context-aware discovery (CaDi) in Alg. 1. CaDi is defined as a recursive algorithm that takes as input an event log, $L$, a pre-defined threshold, $\delta$, and a context-aware process tree (CaT) that is empty initially. The algorithm derives the data-based log split by 2-Means clustering (line 4). In case the quality of the split, as determined by the silhouette coefficient (see Section II-C), satisfies the threshold, we proceed with this split. That is, a constraint operator is derived and added to the CaT (lines 7-8). If the quality is below the threshold, a traditional control-flow-based log split is computed and the respective control-flow operator is added to the CaT (lines 11-12) (see Section II-B). Next, regardless of the applied type of log split (data or control-flow), the obtained sublogs are handled (lines 15-23). That is, we check for the trivial cases of an empty sublog or a sublog with events of a single activity (line 17), and add the respective node to the CaT (line 18). Otherwise, we apply the CaDi algorithm recursively for the sublog and add the result to the tree (line 21). Eventually, the obtained CaT is returned (line 24).

## V. Empirical Evaluation

In this section, we present an empirical evaluation of CaDi. Our main results show that

- CaDi yields higher quality models than a "control-flow first" (and data second) approach when considering both control-flow and context, and,
- CaDi makes less data-oriented errors compared to a "control-flow first" approach.

We first present the experimental setup, before we detail our experimental procedure. Then, we provide a comparative analysis of CaDi against several baselines.

[1]We write $\times$ for the generalized Cartesian product, which, given a set $S = \{s_1, \ldots, s_n\}$, is defined as $\times_{s \in \mathcal{S}} s = s_1 \times \ldots \times s_n$.

### A. Experimental Setup

Below, we define the evaluation measures that we use to assess the quality of a discovered model. Furthermore, we provide an overview of the implementation details for CaDi and the baseline technique to which we compare our algorithm.

*1) Evaluation Measures:* We considered three process quality metrics, each being a function $\psi : \mathcal{L} \times \mathcal{M}_C \to [0, 1]$. Specifically, we measure fitness, $\psi_{Fit}(L, T)$, which evaluates whether a model represents all the behavior recorded in a log, and precision, $\psi_{Pre}(L, T)$, which evaluates whether a model allows for only the behavior recorded in the log [14]. For both measures, we adopt their definition based on multi-perspective alignments [7]. As such, the measures take into account additional perspectives of the event log and in particular event attributes. We transformed the CaT (context-aware process tree) into a Data Petri net (DPN), which contains transitions (modeling activities) that can read and write from and into variables, respectively [15]. Subsequently, we used the multi-perspective process explorer plug-in (ProM[2]) for calculating fitness and precision [16].

We also measure generalization, which is the capability of a discovered model to allow for legal behavior that was not observed in the event log. Following [17], we adopt a well-established method for quantifying the generalization error in machine learning, namely *training and validation* [18]. Specifically, we partition the event log, $L$ into a training set of traces, $L_T$ and a validation set of traces $L_V$. Then, we use the multi-perspective log alignment [7] to align every trace $t \in L_V$ to a model generated based on $L_T$. Let $\gamma$ be the alignment function that maps a trace $t$ and a model $T$ to a real-valued number. Then, the generalization measure is defined as follows:

$$\psi_{Gen}(L, T) = \frac{1}{|L_V|} \sum_{t \in L_V} \gamma(t, T). \tag{3}$$

Lastly, the total quality is a uniform linear combination of fitness, precision, and generalization, namely:

$$\psi_{Tot}(L, T) = \frac{1}{3}\psi_{Fit}(L, T) + \frac{1}{3}\psi_{Pre}(L, T) + \frac{1}{3}\psi_{Gen}(L, T).$$

*2) Implementation:* Our framework was implemented in Python and is publicly available.[3] For the $K$-means implementation, we used the NLTK Python package,[4] which enabled us to replace the default Euclidean distance with the mixed type distance defined in Section II-C.

*3) Baseline:* To evaluate the discovery quality of CaDi, we compared the obtained results to pure control-flow-based discovery with the inductive miner [3] followed by decision mining based on [13]. The latter performs decision point analyses over exclusive choices of the discovered model using decision trees. Here, we used the default ProM parameters with a basic decision tree (no noise filtering) based on minimal fitness. In the remainder, we refer to this baseline as $IM\&DM$.

[2]http://promtools.org

[3]The CaDi algorithm uses *EDU-ProM*, a variant of ProM that allows for non-interactive usage and batch execution [19]. The code in available at https://github.com/dafna-s/Inductive-Context-aware-Process-Discovery

[4]https://www.nltk.org

### B. Datasets and Experimental Procedure

In the experiments, we used three synthetic event logs and a real-world hospital log. The control-flow of our synthetic logs uses a ProM plug-in that generates random process trees and plays out their event logs [20].[5] We used a random process tree, given in Figure 3, to generate a log, $Log0$. It comprises 500 traces, with an average length of 3 events. Next, we enriched $Log0$ with three randomly sampled attributes, namely: time-of-day, resource, and duration.

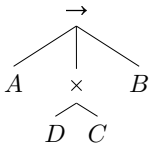



**Fig. 3:** Process tree for Synthetic log ($Log0$).

To challenge our algorithm, we manipulated the attributes of $Log0$ to create three new logs based on the same control-flow. We used truncated exponential sampling [21] to perturb the time-of-day attribute, assuming random case arrivals with exponentially distributed and independent inter-arrival times.[6] Durations and resources were sampled uniformly for each event from a given range and a closed list, respectively. $Log1$, represents a rich log for which we sampled distinct time ranges, resource allocations (with 10% chance for overlapping resources, *e.g.,* activities $A$ and $B$ are performed by '$R1$' and '$R2$' with 0.9 probability and by the other resource with a probability of 0.1, respectively), and activity executions with varying (and potentially overlapping) durations for each process instance. $Log3$ exhibits overlapping time ranges, resource allocations, and durations for all its events. Finally, $Log2$ represents an intermediate log with partial overlapping time ranges, resources and durations.

The real-world hospital log, that we refer to as $Hospital$, comprises a single day of event data from an outpatient cancer hospital in the United States. The dataset comprises about 500 treatment traces that consist of a total of $1,200$ events. Log behavior includes parallelism, loops, and exclusive choices. Regarding data attributes, the log contains a numeric feature that corresponds to time-of-day and two categorical features, namely the resource (*e.g.*, physician, nurse, and physician's assistant) and the department to which the resource belongs.

### C. Results

We begin by analyzing the threshold input $\delta$ that CaDi takes. In what follows, we only focus on $Log2$, since $Log3$ has limited context data and thus little room to tune $\delta$. Furthermore, $Log1$ presents a similar trend to $Log2$. Figure 4 illustrates a sensitivity analysis that presents the multi-perspective quality measures as function of $\delta$.

[5]The synthetic logs are publicly available at https://github.com/dafna-s/Inductive-Context-aware-Process-Discovery/tree/master/Synthetic_log

[6]We assume Poisson arrivals, which is a common arrival scheme in the literature [22]. A truncated distribution is a conditional distribution that restricts the domain of a given distribution.

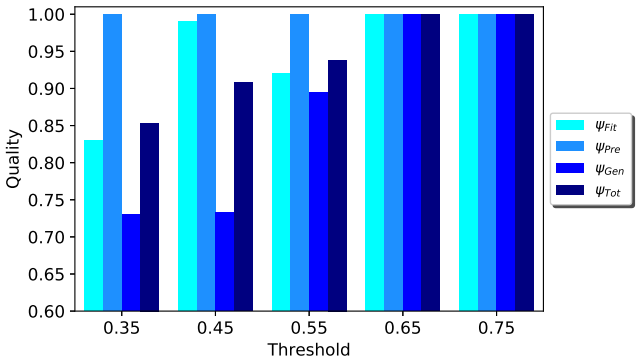



**Fig. 4:** Fitness ($\psi_{Fit}$), Precision ($\psi_{Pre}$), Generalization ($\psi_{Gen}$) and Total Score ($\psi_{Tot}$) by threshold over a synthetic log.

We observe that there is a threshold range where using context improves discovery quality as defined by the three measures. When increasing the threshold beyond 0.75, the algorithm always chooses control-flow over data, leaving out the context perspective. Low threshold levels allow the algorithm to be overly permissive in terms of context, thus reducing control-flow accuracy. For example, setting a .25 threshold for $Log2$ generates over 60 data-based log splits. For the remainder of this section, we choose a threshold value (noted in parenthesis) for each of the logs. This threshold value corresponds to the lowest error rate, chosen via a greedy search, in terms of data and still provides a 2-way data-based split. As an example, Figure 5 illustrates the discovered CaT for $Hospital$ using a threshold of 0.45.

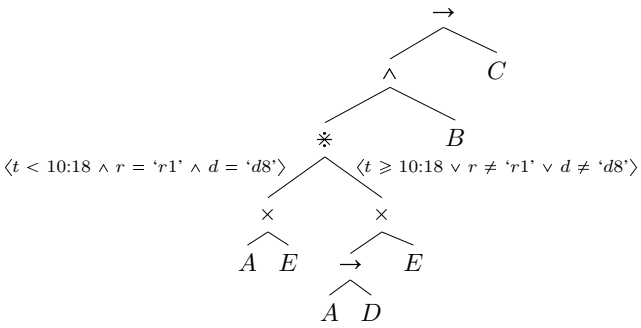



**Fig. 5:** CaT over the $Hospital$ log using CaDi.

Table II reports the overall performance in terms of Fitness ($\psi_{Fit}$), Precision ($\psi_{Pre}$), Generalization ($\psi_{Gen}$) and Total Quality ($\psi_{Tot}$) obtained by CaDi($\delta$) and $IM\&DM$ (see subsubsection V-A3). Clearly, CaDi performs better than $IM\&DM$, on average, with an increase of 14.8% in fitness, 5.4% in precision and 13.6% in generalization, when averaged over the real-world and synthetic datasets. This suggests that analyzing context separation within the discovery yields a better model. Zooming in on the real world dataset, we observe that CaDi has an increased precision compared to $IM\&DM$ with slight improvement in generalization. This emphasizes the ability of

**TABLE II:** Fitness ($\psi_{Fit}$), Precision ($\psi_{Pre}$), Generalization ($\psi_{Gen}$) and Total Quality ($\psi_{Tot}$) over various event logs.

| Dataset | Method | $\psi_{Fit}$ | $\psi_{Pre}$ | $\psi_{Gen}$ | $\psi_{Tot}$ |
|---|---|---|---|---|---|
| *Log1* | CaDi (0.4) | **1.0** | **1.0** | **1.0** | **1.0** |
| | *IM&DM* | 0.993 | **1.0** | 0.977 | 0.99 |
| *Log2* | CaDi (0.75) | **1.0** | **1.0** | **1.0** | **1.0** |
| | *IM&DM* | 0.875 | **1.0** | 0.869 | 0.915 |
| *Log3* | CaDi (0.85) | **1.0** | **1.0** | **1.0** | **1.0** |
| | *IM&DM* | 0.75 | **1.0** | 0.75 | 0.833 |
| *Hospital* | CaDi (0.45) | **0.917** | **0.957** | **0.868** | **0.914** |
| | *IM&DM* | 0.826 | 0.785 | 0.838 | 0.816 |

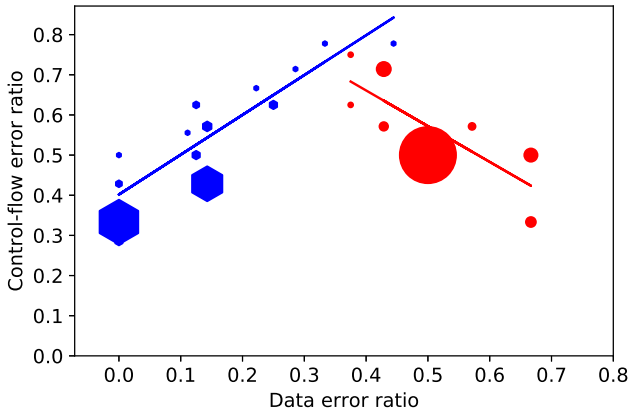



**Fig. 6:** Comparison of CaDi (blue) and $IM\&DM$ (red) in terms of data ($x$-axis) and control-flow ($y$-axis) deviations over the real-world dataset. Larger markers indicate higher frequency of the respective values.

CaDi to create a model that better describes the observed and unobserved behavior of the business process.

We conclude by analyzing misalignment types of models that result from $IM\&DM$ and CaDi. Figure 6 presents, for each of the discovered models, the ratio of data and control-flow misalignments for every trace of the real-world dataset, *i.e.*, the number of traces that were misaligned using the model out of all traces. Note that larger markers indicate higher frequencies of the respective value. We observe that CaDi makes less data-oriented errors compared to $IM\&DM$. This shows the ability of CaDi to capture the data perspective during discovery. Furthermore, an increase in data deviations correlates with an increase in control-flow deviations. We note that this may be the product of CaDi breaking the control-flow based on context (dissatisfying constraints, thus leading to control-flow deviations). Moreover, we observe that $IM\&DM$ has fewer control-flow deviations, which corresponds to its "control-flow first" approach. CaDi, on the other hand, compensates for control-flow misalignments with context splits thus yielding quality improvement in the discovered models.

## VI. Related Work

Process discovery (see [23] for a review) has been an active research area for many years. The inductive miner is one of the well-established discovery algorithms, guaranteeing soundness and fitness for the discovered models [3]. Our proposed algorithm follows the divide-and-conquer approach of the inductive miner, as it also splits the event log, and performs an inductive discovery step on the resulting log splits. CaDi, however, goes beyond the inductive miner and considers data attributes to discover the context of activity execution.

Previous work on context in process discovery mostly prioritized control-flow over data. Specifically, as first step, a control-flow model (e.g., a Petri net) is discovered. In a second step, the model is annotated with decisions [13], [24], resources and time [25], queues [22], and general context information [26]. These approaches improve performance analysis and predictive monitoring using the additional context in the enhancement stage, whereas in our work, we focus on improving the quality of the resulting model at the discovery stage. Therefore, we measure "success" of our context-aware algorithm by measuring multi-perspective alignments.

Context information was also considered in other declarative and procedure discovery methods. In [27] and [28], the authors propose techniques for mining declarative models with conditions and discovering constraints that include data attributes, respectively. Further, the recently proposed data-aware heuristic miner is a procedural method that filters infrequent paths based on values of data attributes [29]. The latter produces control-flow models that do not represent the underlying context that led to their discovery. The main difference with our work is that these papers take a control-flow first perspective, while our work considers the two perspectives in combination.

In a recent work, context was represented in the form of temporal networks to discover both the control-flow and performance related perspectives [30], simultaneously. These graphs are designated to capture temporal aspects (e.g., time-of-day) and time-varying congestion (e.g., queueing). CaDi aims to discover general contexts and is not limited to the performance dimension of a process.

Our work is also related to discovery based on trace clustering. Related approaches comprise two steps: (i) traces of an event log are clustered based on a pre-defined similarity measure, and (ii) a model is derived for each of the clustered sublogs. Common trace clustering methods include [31], [32], [33], [34], [35]. Recently, trace clustering has also been conducted such that the impact of clustering on discovery is taken into account [36]. Related ideas have been presented for slicing and dicing of discovered models [37]. Specifically, the method splits-up models by trace clustering, and dices models by identifying hierarchical subprocesses. The main difference to our work is that trace clustering allows only for horizontal log split, while in our work, we split the log both horizontally and vertically.

## VII. Conclusions and Future Work

In this work, we proposed an algorithm for context-aware process discovery (CaDi), which considers both control-flow and context, simultaneously. The algorithm discovers context-aware process trees, which we define as a suitable representational bias to capture both control-flow and data perspectives. Specifically, CaDi combines inductive mining and $K$-means clustering to generate process models that capture context information explicitly through a constraint operator.

Using multi-perspective alignments we validate the superiority of CaDi, compared to an inductive miner discovery followed by a decision point analysis. The latter approach represents a "control-flow first" method, which prioritizes control-flow over data. We further analyze the conformance errors of the models and show that while CaT is somewhat less accurate in terms of control-flow, it shows much less data-oriented errors compared to "control-flow first" methods.

In future work, we plan to extend context-aware discovery to involve attentional machine-learning techniques to analyze the events' contexts, for example using embeddings [38], [39]. Furthermore, we wish to develop context-aware evaluation measures that assess the quality of a given context-aware process tree. That includes replay semantics for CaTs that incorporate general constraints rather than transforming them into an activity-specific guard.

In addition, we intend to explore understandability issues when handling rich event logs. We aim at balancing between data splits that consider numerous features and data splits that use less features to increase understandability of context splits.

## References


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

[2] W. Van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 9, pp. 1128–1142, 2004.

[3] S. J. Leemans, D. Fahland, and W. M. van der Aalst, "Discovering block-structured process models from event logs-a constructive approach," in *ATPN*. Springer, 2013, pp. 311–329.

[4] F. M. Maggi, C. D. Francescomarino, M. Dumas, and C. Ghidini, "Predictive monitoring of business processes," in *CAiSE*. Springer, 2014, pp. 457–472.

[5] J. Carmona, B. van Dongen, A. Solti, and M. Weidlich, *Conformance Checking: Relating Processes and Models*. Springer, 2018.

[6] A. Senderovich, M. Weidlich, L. Yedidsion, A. Gal, A. Mandelbaum, S. Kadish, and C. A. Bunnell, "Conformance checking and performance improvement in scheduled processes: A queueing-network perspective," *Inf. Syst.*, vol. 62, pp. 185–206, 2016.

[7] F. Mannhardt, M. De Leoni, H. A. Reijers, and W. M. Van Der Aalst, "Balanced multi-perspective checking of process conformance," *Computing*, vol. 98, no. 4, pp. 407–437, 2016.

[8] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering block-structured process models from event logs containing infrequent behaviour," in *BPM Workshops*. Springer, 2013, pp. 66–78.

[9] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the 5th Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. USA, 1967, pp. 281–297.

[10] R. Xu and D. Wunsch, "Survey of clustering algorithms," *IEEE TNN*, vol. 16, no. 3, pp. 645–678, 2005.

[11] A. Ahmad and L. Dey, "A k-mean clustering algorithm for mixed numeric and categorical data," *Data & Knowledge Engineering*, vol. 63, no. 2, pp. 503–527, 2007.

[12] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.

[13] A. Rozinat and W. M. P. Aalst, *Decision mining in business processes*. Beta, Research School for Operations Management and Logistics, 2006.

[14] J. C. Buijs, B. F. van Dongen, and W. M. van der Aalst, "Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity," *Int'l Journal of Cooperative Inf. Sys.*, vol. 23, no. 01, p. 1440001, 2014.

[15] M. De Leoni and W. M. van der Aalst, "Data-aware process mining: discovering decisions in processes using alignments," in *ACM SAC*. ACM, 2013, pp. 1454–1461.

[16] F. Mannhardt, M. De Leoni, and H. A. Reijers, "The multi-perspective process explorer." *BPM (Demos)*, vol. 1418, pp. 130–134, 2015.

[17] Y. Dahari, A. Gal, A. Senderovich, and M. Weidlich, "Fusion-based process discovery," in *CAiSE*. Springer, 2018, pp. 291–307.

[18] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. Springer series in statistics, 2001, vol. 1, no. 10.

[19] Y. Dahari, A. Gal, and A. Senderovich, "Edu-prom: Prom for the classroom," in *BPM (Demos)*, 2017.

[20] T. Jouck and B. Depaire, "Simulating process trees using discrete-event simulation," 2017.

[21] K. Balakrishnan, *Exponential distribution: theory, methods and applications*. Routledge, 2018.

[22] A. Senderovich, M. Weidlich, A. Gal, and A. Mandelbaum, "Queue mining–predicting delays in service processes," in *CAiSE*. Springer, 2014, pp. 42–57.

[23] A. Augusto, R. Conforti, M. Dumas, M. L. Rosa, F. M. Maggi, A. Marrella, M. Mecella, and A. Soo, "Automated discovery of process models from event logs: Review and benchmark," *CoRR*, vol. abs/1705.02288, 2017.

[24] J. D. Smedt, F. Hasic, S. K. L. M. vanden Broucke, and J. Vanthienen, "Towards a holistic discovery of decisions in process-aware information systems," in *BPM*, 2017, pp. 183–199.

[25] A. Rozinat, R. S. Mans, M. Song, and W. M. van der Aalst, "Discovering simulation models," *Inf. Sys.*, vol. 34, no. 3, pp. 305–327, 2009.

[26] W. M. P. van der Aalst, M. Schonenberg, and M. Song, "Time prediction based on process mining," *Inf. Sys.*, vol. 36, no. 2, pp. 450–475, 2011.

[27] F. M. Maggi, M. Dumas, L. García-Bañuelos, and M. Montali, "Discovering data-aware declarative process models from event logs," in *BPM*. Springer, 2013, pp. 81–96.

[28] V. Leno, M. Dumas, and F. M. Maggi, "Correlating activation and target conditions in data-aware declarative process discovery," in *BPM*, 2018, pp. 176–193.

[29] F. Mannhardt, M. de Leoni, H. A. Reijers, and W. M. P. van der Aalst, "Data-driven process discovery - revealing conditional infrequent behavior from event logs," in *CAiSE*. Springer, 2017, pp. 545–560.

[30] A. Senderovich, M. Weidlich, and A. Gal, "Temporal network representation of event logs for improved performance modelling in business processes," in *BPM*. Springer, 2017, pp. 3–21.

[31] D. R. Ferreira, M. Zacarias, M. Malheiros, and P. Ferreira, "Approaching process mining with sequence clustering: Experiments and findings," in *BPM*. Springer, 2007, pp. 360–374.

[32] R. P. J. C. Bose and W. M. P. van der Aalst, "Context aware trace clustering: Towards improving process mining results," in *SDM*. SIAM, 2009, pp. 401–412.

[33] G. Greco, A. Guzzo, L. Pontieri, and D. Saccà, "Discovering expressive process models by clustering log traces," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 8, pp. 1010–1027, 2006.

[34] J. D. Weerdt, S. K. L. M. vanden Broucke, J. Vanthienen, and B. Baesens, "Active trace clustering for improved process discovery," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 12, pp. 2708–2720, 2013.

[35] T. Chatain, J. Carmona, and B. F. van Dongen, "Alignment-based trace clustering," in *ER*. Springer, 2017, pp. 295–308.

[36] Y. Sun, B. Bauer, and M. Weidlich, "Compound trace clustering to generate accurate and simple sub-process models," in *ICSOC*. Springer, 2017, pp. 175–190.

[37] C. C. Ekanayake, M. Dumas, L. García-Bañuelos, and M. L. Rosa, "Slice, mine and dice: Complexity-aware automated discovery of business process models," in *BPM*. Springer, 2013, pp. 49–64.

[38] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *NIPS*, 2013, pp. 3111–3119.

[39] P. De Koninck, S. vanden Broucke, J. De Weerdt, "act2vec, trace2vec, log2vec, and model2vec: Representation Learning for Business Processes," in *BPM*. Springer, 2018, pp. 305–321.

# Prime Miner - Process Discovery using Prime Event Structures

Robin Bergenthum
*Fakultät für Mathematik und Informatik*
*FernUniversität in Hagen, Germany*

***Abstract*—We present a new region-based approach to process discovery using prime event structures as an intermediate model. We use the prime event structure to generate precise models of the most frequent use-cases captured by an event-log. We start with an event log, apply a concurrency oracle to construct a partial language, and fold the language into prime event structures. We apply the theory of compact tokenflow regions to these structures to synthesize a sequence of Petri nets representing the most frequent partially ordered runs of the recorded behavior. The sequence of Petri nets has increasing fitness but decreasing precision. To highlight the benefits of such an approach, we introduce a plug-in for the tool ProM called *Prime Miner*, implementing the new concepts.**

## 1. Introduction

Process mining [3,20] is analyzing recorded behavior of a business process to gain knowledge about performance and conformance of the process at hand. In the past decade, a high variety of process mining algorithms and methods have been introduced. Furthermore, there are a large number of academic and commercial tools, contests, and case studies. The homepage http://www.win.tue.nl/ieeetfpm of the IEEE Task Force on Process Mining provides a well-sorted introduction.

Process mining is based on two formalisms: *event logs* and *workflow models*. An event log is a sequence of events recording behavior of a business process. A workflow model is an executable, often Petri net-like, model of a business process. Usually, all process mining operations are defined using these two notions. However, in recent years, a new type of approaches have been arising which tackle process mining operations using partially ordered representations of behavior. These are, for example, process discovery using causal runs [18] or unfoldings [24], model simplification based on branching processes [19], conformance checking using partially ordered traces [22], and model comparison using event structures [6]. This trend continued up to the point when Marlon Dumas and Luciano García-Bañuelos suggested recasting all process mining operations based on *prime event structures* [15]. A prime event structure [23] is a set of events just like an event log; yet, these are not ordered in a sequence, but in a partial order. Thus, using prime event structures, we are able to explicitly express concurrency. Paper [15] suggests that *prime event structures* serve as a perfect link between *event logs* and *workflow models*.

Primus inter pares of all process mining operations is *process discovery* [2,4]. Let there be some event log recording behavior of a business process, then process discovery is to generate a workflow model which is likely to be a good representation of the business process. Some of the outstanding candidates, each representing a whole class of different discovery algorithms, are: the famous $\alpha$-algorithm [4], the heuristics-miner [26], discovery using state-based regions [14], discovery with regions of languages [12,13,27], folding of partial orders [11,17], and the inductive miner [21]. For more details, we refer the reader to [3,20].

Every discovery algorithm applies its own ideas and techniques to progress from the recorded behavior to a process model. Yet, they all have in common that, by doing so, they need to fulfill four goals: (1) go from the level of events to the level of actions, (2) induce a concurrency relation on the level of events or actions, (3) deal with *noise*, (4) find a reasonable balance between *fitness* and *precision*. For most discovery techniques, these four goals are achieved in one or two big steps. Following the line of thought of [15], we can fine-tune and increase the transparency of a discovery algorithm by making smaller steps and using an intermediate model. Encouraging examples of discovery algorithms applying this concept and even predating the more general suggestions proposed in [15] are approaches that *fold* representations of a partial language into an integrated process model [11,17,18,24]. Roughly speaking, *folding* is merging events that carry the same label. In some sense, folding produces the product of all choices observed in the control-flow of the workflow process, thus, it is a quite brutal generalization of the recorded behavior. However, after such folding, the event structure can easily be transformed into a workflow model. Paper [24] tackles the drawbacks of brute force folding by adding a set of negative traces specified by domain experts and allowing splitting of labels to get more precise results.

In this paper, we present a new region-based discovery algorithm we call the *Prime Miner*. The *Prime Miner* follows the ideas of [15] and [24] but instead of *folding*, we use *synthesis* to generate a model from the prime event structure. We link concepts from event structure based and region-based mining to present a new approach with a lot of control over the calculated results. Using synthesis, we obtain an approach tailored to applications which require models with high precision.

Well-known discovery algorithms using synthesis for process discovery are [13,27] and [14]. These approaches

978-1-7281-0919-0/19/$31.00 ©2019 European Union
DOI 10.1109/ICPM.2019.00017

produce models with perfect *fitness* and best possible *precision* in the chosen semantic and net class. Yet, this comes at a high cost: region-based approaches are known to have *poor run-time* and to produce quite *complex models*. The main idea of the *Prime Miner* is to use prime event structures to minimize the troubles of region-based approaches when applied to process discovery. We do not synthesize from the language or the state graph of the event log but use a *concurrency oracle* to build a prime event structure first. Less input into the synthesis leads to a better *run-time*. A prime event structure models a set of use-cases of a business process. We can choose a subset of the most frequent use-cases to synthesize our model. Here, we can sacrifice *fitness* for *run-time*. We apply a new region definition tailored to partial languages and prime event structures called *compact tokenflow regions* [7]. For these structures, it is the most efficient region definition yet. For this contribution, we restrict *compact tokenflow regions* to one-safe nets. Synthesizing general Petri nets is more precise but much slower. Here, we sacrifice *precision* for *readability* and *run-time*.

The *Prime Miner* is an enhanced and improved version of the discovery algorithm *Eve*. *Eve* was presented in the short workshop-paper [8]. We implement the Prime Miner as a plug-in for the most prominent process mining tool ProM [16]. The remainder of the paper is structured as follows: we introduce the *Prime Miner* in Section 2. This section is structured following the main steps of the *Prime Miner*: we (a) calculate the language of an event log, (b) apply a concurrency oracle to mine a concurrency relation, (c) fold the language into a partial language using the concurrency relation, (d) fold the partial language into a prime event structure, (e) synthesize a workflow model adapting the theory of regions to prime event structures, and (f) find a good balance between fitness and precision. We show experimental results in Section 3. Section 4 concludes the paper.

## 2. The Prime Miner

We assume the reader is familiar with Petri nets and the basic concepts of process discovery (see for example [3,20,25] for well-sorted introductions). This section presents our new discovery approach we called the *Prime Miner*. To showcase the different steps, we provide a well-known running example. The example is the *repair example* taken from the ProM framework tutorial (see www.processmining.org/).

### (a) Calculate the Language of an Event Log

The input for every process discovery algorithm is an event log. An event log is a sequence of events. Each event records an action performed in a business process. Thus, every case relates to a trace of actions. We formalize the event log by a multi-set of traces. The language of an event log is the set of maximal traces of the support of its multi-set. The first step of the Prime Miner is to calculate the language of an event log.

**Definition 1.** (Event Log)
*Let $T$ be a finite set of actions. A trace $\sigma$ is a sequence of actions, i.e. $\sigma \in T^*$. An event log $L$ is a multi-set of traces of $T$, i.e. $L \in \mathbb{B}(T^*)$. We call the set of maximal traces of the support of $L$ the language of the event log. We denote the language of an event log $L$ by $\mathcal{L}(L)$.*

In the *repair example*, the set of actions is {*Register*, *Analyze Defect*, *Repair (Complex)*, *Repair (Simple)*, *Test Repair*, *Inform User*, *Restart Repair*, *Archive Repair*}. The event log records $11855$ events, $1104$ cases, and $39$ different traces in the language of the log.

### (b) Mine a Concurrency Relation

The second step of the Prime Miner is to deduce a concurrency relation from an event log to merge the language into a set of partial orders. This is to explicitly distinguish between conflicting and concurrent actions recorded from the business process.

In every business process, for every case, there is an implicit set of choices to be made that direct the control-flow of actions. In the *repair example*, there is the choice whether the repair is complex or simple. Thus, there is a conflict between cases executing the *Repair (Complex)* and the *Repair (Simple)* action. Obviously, the related cases in the event log will record different sequences of actions. If we consider the two actions *Inform User* and *Repair (Complex)* in the same example, almost every discovery method will assume both actions to be executable concurrently. Obviously, there is no real choice regarding these two actions, they just occur in any order. Consequently, related traces may look different but record the same set of choices, i.e., the same *run* of the process.

The Prime Miner identifies runs of the event log using so-called *concurrency oracles* (see [5] for an in-depth discussion of different oracles). We need a concurrency oracle because it is impossible to directly observe concurrency of events. The Prime Miner implements two different concurrency oracles to choose from during the mining procedure. Both oracles detect concurrency on the level of actions, i.e., if two actions are concurrent, the related events are concurrent in every case. Using prime event structures, it is also possible to specify concurrency at the level of events but for most practical examples this is overkill and often leads to models that are very precise but lack *generalization* and *readability*. We refer the reader to [5] for a more detailed discussion.

The first oracle implemented in the Prime Miner is the life-cycle oracle. This oracle requires additional event data. We apply the life-cycle oracle as soon as there is life-cycle information attached to the events of an event log. If the lifespans of two events of the same case intersect, we deduce a concurrency relation. If we see the same relation often enough in the event log, we assume the actions to be concurrent.

The *repair example* includes life-cycle information. For every executed action, there are two events in the event log; one event labeled with start, and a second labeled with

complete. There are many cases in this event log where we see the start events of *Inform User* and *Repair (Complex)* before we see the related complete events, i.e., their lifespans intersect. Here, the Prime Miner deduces a concurrency relation on the level of actions. Altogether, using the Prime Miner, we obtain (*Repair (Simple), Inform User*), (*Repair (Complex), Inform User*), and (*Test Repair, Inform User*) as concurrent pairs of actions in our example. Note that most mining algorithms will also assume (*Restart Repair, Inform User*) to be concurrent. However, in the repair example log file, the lifespans of none of the occurrences of the two actions intersect in any case. This can be some useful information when dealing with the repair process. In that sense, we consider the life-cycle oracle to be quite precise.

The second oracle implemented in the Prime Miner is the $\alpha$-*oracle*. This oracle requires no additional data. As suggested in paper [15], the conflict and concurrency matrix of the $\alpha$-algorithm [4] is used to define a concurrency relation on the level of actions. We assume actions to be concurrent if related events occur as direct predecessors in arbitrary order. Since the $\alpha$-oracle does not need any additional event data, we can directly apply it to the language of the event log. In the *repair example*, we receive (*Repair (Simple), Inform User*), (*Repair (Complex), Inform User*), (*Test Repair, Inform User*), and (*Restart Repair, Inform User*) as concurrent pairs of actions. Other than the life-cycle oracle, the $\alpha$-oracle assumes concurrency for *Restart Repair* and *Inform User*. In that sense, we consider the $\alpha$-oracle the more liberal oracle of the Prime Miner.

**(c) Fold the Language into a Partial Language**

The third step of the Prime Miner is to calculate a partial language from the language of the event log using the concurrency relation calculated in the previous step. This is to examine the runs hidden in the event log. To explicitly model runs, we use the notion of labeled Hasse diagrams.

**Definition 2.** (Labeled Hasse Diagram)
*Let $T$ be a finite set of actions. A labeled partial order is a triple $lpo = (V, <, l)$ where $V$ is a finite set of nodes, $< \subseteq V \times V$ is a transitive and irreflexive relation, and the labeling function $l : V \to T$ assigns a action to every node. We denote $<^*$ the transitive closure of $<$ and $<^\diamond$ the transitive reduction of $<$.*
*A triple $run = (V, <, l)$ is a labeled Hasse diagram iff $(V, <^*, l)$ is a labeled partial order and $<^\diamond = <$ holds.*

For more details on how to specify behavior in terms of Hasse diagrams, we refer the reader to [7].

In this paper, we depict Hasse diagrams in terms of labeled Petri nets. Every event is depicted by a transition; every arc is depicted by a place. Every place is connected like the related arc in the Hasse diagram. Thus, the flow-relations match. A Petri net depicting a Hasse diagram is labeled and will always be conflict-free, because every place will have exactly one ingoing and one outgoing arc. Other common terms for these structures are: *marked graphs*, *occurrence nets*, or *causal runs*. Whatever we call the set of Hasse diagrams, they are just partially ordered sets of events.

Figure 1 depicts the sequence *Register*, *Analyze Defect*, *Inform User*, *Repair (Complex)*, *Test Repair*, *Archive Repair* as a Hasse diagram. This sequence of actions appears 328 times in the *repair example* event log. Roughly speaking, every third case records this behavior.

The next step of the Prime Miner is to apply the concurrency relation calculated in the last step. For every sequence of actions in the language of the event log, we build the transitive closure, delete all arcs relating to concurrent actions, and calculate the transitive reduction to generate a labeled Hasse diagram representing the concurrency relation mined by the concurrency oracle.

**Definition 3.** (Partial Language of an Event Log)
*Let $T$ be a finite set of actions, $L$ be an event log, and $\Gamma \subseteq T \times T$ be a concurrency relation.*
*We define the partial language of $L$ by $\mathcal{P}_\Gamma(L) = \{(V, >, l) \mid e_1 \dots e_n \in \mathcal{L}(L), V = \{e_1, \dots, e_n\}, <= \{(e_i, e_j) \mid 1 \leq i < j \leq n, (l(e_i), l(e_j)) \notin \Gamma\}^\diamond\}$.*

In the *repair example*, the life-cycle oracle and the $\alpha$-oracle assume (*Repair (Complex), Inform User*) and (*Test Repair, Inform User*) to be concurrent. If we consider the sequence of actions in Figure 1, the related Hasse diagram generated by the Prime Miner is depicted in Figure 2. Using the $\alpha$-oracle, the partial language of the *repair example* event log folds to only 9 different labeled Hasse diagrams. Remember, the event log has 1104 cases relating to 39 different sequences of actions. So, having 9 runs only is already a massive reduction. Furthermore, there are 524 cases relating to the Hasse diagram depicted in Figure 3. In that sense, Figure 3 is already a notable process model of the *repair example*. Figure 3 depicts a single run only, but is able to replay over 50% of the observed behavior.

**(d) Fold Partial Language to a Prime Event Structure**

The next step of the Prime Miner is to merge the partial language into a prime event structure. This is a preparatory step before we start synthesizing the final process models. As mentioned in the introduction, we have to minimize the input to the synthesis part of the Prime Miner as much as possible to keep the overall run-time of the synthesis in check.

**Definition 4.** (Labeled Prime Event Structure)
*Let $T$ be a finite set of actions. A labeled prime event structure is a tuple $(V, <, \#, l)$ where $(V, <, l)$ is a Hasse diagram and $\# \subseteq V \times V$ is an irreflexive, symmetric relation satisfying $e \# e' \wedge e' <^* e'' \Rightarrow e \# e''$.*

A labeled prime event structure is a labeled Hasse diagram with an additional conflict relation. Thus, a labeled prime event structure is able to express multiple runs at once. Just like a Hasse diagram, an event structure is able to explicitly express concurrency between events. Two events occur concurrently if they are neither ordered by the $<$-relation nor ordered by the $\#$-relation. Maximal sets of events, where no pair is in the $\#$-relation, are so-called consistency sets.

The Prime Miner folds a partial language of the event log into a single prime event structure so that the set of all

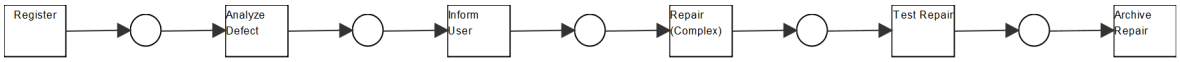



Figure 1. Hasse diagram of the most frequent sequence recorded in the repair example.

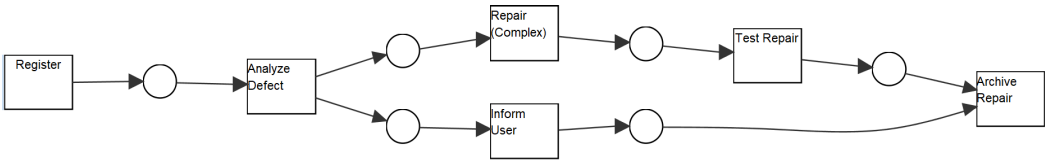



Figure 2. Hasse diagram of the most frequent run recorded in the repair example.

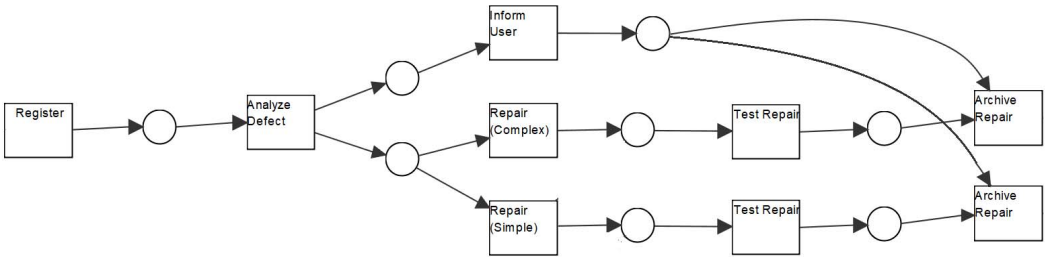



Figure 3. Prime event structure of the two most frequent runs of the repair example.

Hasse diagrams is the set of consistency sets. First, we copy all Hasse diagrams into a single yet unmerged prime event structure, so that every node relates to one consistency set only and two nodes are in conflict if they relate to different Hasse diagrams. Now, if prefixes of different consistency sets are isomorphic, we merge them. We merge the nodes of these prefixes, but keep the conflict relations for all nodes after the common prefix. If we forbid auto-concurrency in the Hasse diagrams, the resulting prime event structure is unique. After this procedure, the set of consistency sets is still the set of Hasse diagrams but the structure is much more compact.

As we did for Hasse diagrams, we depict labeled prime event structures as labeled Petri nets in this paper. This is very convenient because we can display the conflict relation in terms of conflict places. If two events are in conflict, the related transitions are in conflict as well. If two transitions are in conflict, all later transitions are in conflict as well. Note that another well-known term for these structures is *branching process*. Figure 3 depicts a prime event structure of our running example. The most frequent run of the *repair example* is depicted in Figure 2. The second most frequent run is the same run, but replaces the event labeled *Repair (Complex)* with an event labeled *Repair (Simple)*. If we fold these two Hasse diagrams, we obtain the labeled prime event structure of Figure 3. *Register*, *Analyze Defect*, and *Inform User* is the common prefix. There is a conflict between *Repair (Complex)* and *Repair (Simple)* and this conflict is passed on to later events accordingly. The two consistency sets of the labeled prime event structure of Figure 3 are the two most frequent runs of our example. Altogether, this structure is able to replay almost 75% of all cases of the *repair example* event log. Of course, if we merge all 9 runs of this example into a prime event structure, we are able to replay all of the traces. However, there is still a lot of unnecessary splitting of labels.

**(e) Synthesize a Net from the Prime Event Structure**

The next step of the Prime Miner is to synthesize a process model from the labeled prime event structure using the theory of regions to obtain a fitting and precise integrated model.

Every region-based synthesis approach is based on some definition of the language of a Petri net. Since the semantic of the Prime Miner is based on a set of runs, i.e., a set of Hasse diagrams or a prime event structure recording behavior of our business process, we need an appropriate characterization of the language of a Petri net. A Hasse diagram is in the language of a Petri net if there are so-called compact tokenflows describing valid distributions of tokens along the arcs of such a diagram for every place of the net [9,10].

**Definition 5.** (Valid Hasse Diagram)
*Let $(P, T, W, m_0)$ be a marked p/t-net and $(V, <, l)$ be a labeled Hasse diagram so that $l(V) \subseteq T$ holds.*
*A compact tokenflow is a function $x : (V \cup <) \to \mathbb{N}$.*
*$x$ is valid for $p \in P$ iff the following conditions hold:*

$$\forall\, v \in V : x(v) + \sum_{v' < v} x(v', v) \geq W(p, l(v)) \text{ and}$$

$$\sum_{v < v'} x(v, v') \leq x(v) + \sum_{v' < v} x(v', v) - W(p, l(v)) + W(l(v), p).$$

$$\text{As well as: } \sum_{v \in V} x(v) \leq m_0(p).$$

*$(V, <, l)$ is valid for $(P, T, W, m_0)$ iff there is a valid tokenflow for every $p \in P$.*

Every node of the Hasse diagram models the occurrence of a transition. If we fix a place, every occurrence of a transition will consume and produce some fixed number of tokens. A tokenflow tells the story of a valid distribution of tokens if it satisfies all conditions of Definition 5: first, every node receives enough tokens, second, no node has to pass too many tokens, and third, the initial marking is not exceeded. Altogether, the language of a marked Petri net is well-defined by the set of valid labeled Hasse diagrams [9,10].

For the Prime Miner, we lift the notion of valid tokenflows to prime event structures and restrict every tokenflow to be one-safe. We define a tokenflow on a labeled prime event structure as valid if its projection onto every consistency set is valid for the related Hasse diagram. Thus, a valid tokenflow on a prime event structure directly defines a set of valid tokenflows for the related Hasse diagrams.

**Definition 6.** (Valid One-Safe Compact Tokenflow)
*Let $(P, T, W, m_0)$ be a marked p/t-net, $(V, <, \#, l)$ be a labeled prime event structure with the set of consistency sets $C = \{V_1, \ldots, V_n\}$ so that $l(V) \subseteq T$ holds. A one-safe compact tokenflow is a function $x : (V \cup <) \to \{0, 1\}$.*
*$x$ is valid for $p \in P$ iff $\forall\ V_i \in C : x|_{(V_i \cup (V_i \times V_i))}$ defined on the Hasse diagram $(V_i, <|_{(V_i \times V_i)}, l|_{V_i})$ is valid for $p$.*

Roughly speaking, every place with a related valid one-safe tokenflow on a labeled prime event structure does not prohibit the execution of any modeled consistency set.

In the Prime Miner, we are now able to start the synthesis procedure. We start with an empty Petri net, add a transition for every action of a labeled prime event structure, and calculate a set of places to connect the set of transitions. We only add places so that there is a one-safe valid tokenflow for that place. Thus, every consistency set will be executable by the generated net and we will have perfect fitness. Now we only need to characterize the set of all places that relate to valid one-safe compact tokenflows and introduce the notion of one-safe compact tokenflow regions.

**Definition 7.** (One-Safe Compact Tokenflow Region)
*Let $(V, <, \#, l)$ be a labeled prime event structure so that $l(V) \subseteq T$ holds and $p$ be a place.*
*$r : ((V \cup <) \cup (T \times \{p\}) \cup (\{p\} \times T) \cup \{p\}) \to \{0, 1\}$ is a one-safe compact tokenflow region for $(V, <, \#, l)$ iff $r$ is valid for $p$ in $(\{p\}, T, r|_{(T \times \{p\}) \cup (\{p\} \times T)}, r(p))$.*

Every region defines a feasible place for the Petri net we synthesize. Like in any region-based approach, it is sufficient to add places related to minimal regions. We construct the set of minimal regions using the same methods applied for state-based regions. We calculate all excitation- and switching-regions to have a set of so called pre-regions (see [14] for more details) and check if every pre-region is a region. For every region, we add the related place to the net. For all others, we expand each pre-region to construct a new set of pre-regions. In state-based region theory, we expand pre-regions by adding states so that the gradients of equally labeled transitions align. Here, we expand pre-regions by adding tokenflow to arcs of the prime event structure so that the conditions of Definition 5 for the nodes of the consistency sets align. Finally, we will obtain a one-safe Petri net that can execute all consistency sets of the prime event structure. Furthermore, this net is the best upper approximation to the partial language defined by the labeled prime event structure in the net class of one-safe Petri nets. For more details, we refer the reader to [14] and [7].

**(f) The Final Touch**

The last step of the Prime Miner is to find a good balance between fitness and precision for the actual result of the mining procedure. This is not a single step but a loop of steps creating a sequence of different process models with increasing fitness and decreasing precision and simplicity.

The reason we add such a step to the Prime Miner is that region-based mining techniques tend to generate process models with perfect fitness but lacking precision. A region-based approach will always be able to execute all traces, Hasse diagrams, and prime event structures input into the synthesis procedure. If an event log contains noise, the resulting process models tend to be hardly readable for practical examples containing quite complex structures using heavily branching places. In the Prime Miner, we take an iterative approach based on frequencies of runs of the event log to tackle these problems of synthesis-based approaches.

First, we take the Hasse diagram that relates to most cases in the event log to synthesize a Petri net. This is the first result of the Prime Miner. Then, we add the second most frequent Hasse diagram to the first to build a labeled prime event structure and use this structure to synthesize a second result. We proceed with the third most frequent Hasse diagram, build another prime event structure, and synthesize the next result, altogether synthesizing a sequence of Petri net models. We go on like this until we have considered all the Hasse diagrams.

For every generated model there are three options: (1) it is a well-connected, new Petri net and we add this Petri net to the set of results, (2) the generated Petri net was already generated in a previous iteration and we do not need to add this Petri net again, (3) the Petri net is not connected properly, meaning that there are transitions with an empty preset. Here, the region approach could not calculate a fitting region and we have a reason to assume that the recorded behavior does not match the observed process well. We discard the Hasse diagram and do not consider it again in later iterations.

In the *repair example*, we start this phase of the Prime Miner with 9 Hasse diagrams. The frequencies, i.e., matching numbers of cases in the event log of these diagrams are: 524, 211, 101, 80, 36, 26, 11, 9, 2. The Prime Miner synthesizes 3 Petri nets. The first net is able to execute the first Hasse diagram, the second net is able to execute the first two Hasse diagrams, and the third net is able to execute all nine Hasse diagrams. In this example, we obtain the same result from folding and synthesizing the first three Hasse diagrams and from folding and synthesizing all of them. In this sense, the repair example does not include any noise because even the infrequent runs match the process model of the frequent ones.

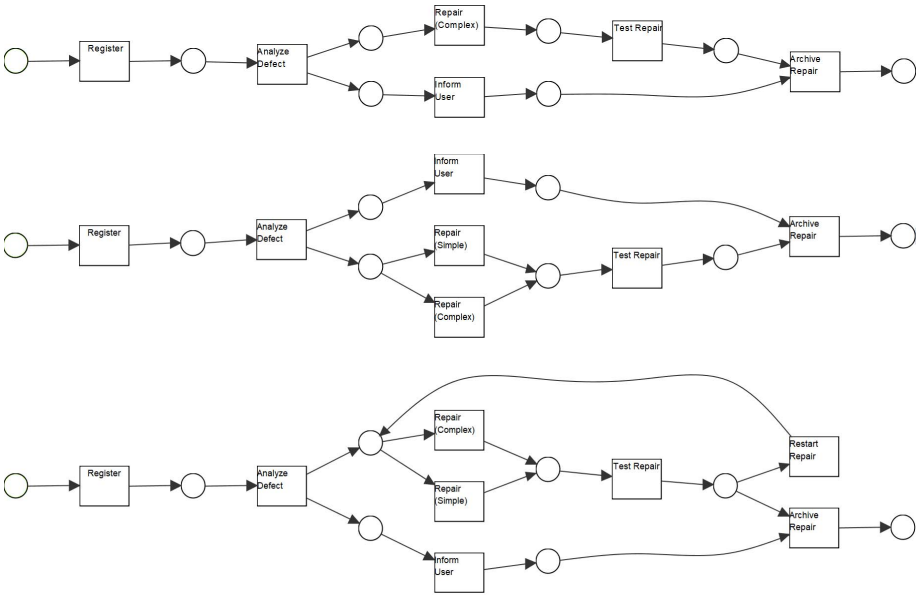



Figure 4. Petri nets of the repair example.

Figure 4 depicts all three Petri nets synthesized in the *repair example*. The first Petri net directly relates to the most frequent run of the business process. It is able to execute $52.0\%$ of all cases of the event log. The model is precise and simple. The second Petri net relates to the alternative between the two most frequent runs, one executing the simple and one executing the complex repair. This model is still simple and precise and is able to execute $73.0\%$ of all cases. The third Petri net includes the first two runs and all the looping behavior of runs three to nine as well. This model has perfect fitness. Note that we restrict the Prime Miner to one-safe Petri nets in step (e), thus, the loop can be executed repeatedly. Altogether, this model is less precise than the first two.

Summing up, in the *repair example*, the Prime Miner generates three appropriate models. The sequence of models reflects the set of runs underlying the event log. We obtain not only process models of the control flow but also information about frequent and infrequent behavior based on the notion of prime event structures. The frequencies are considered on the level of runs, not only on the level of traces. The sequence of models has increasing fitness but decreasing precision.

## 3. Implementation and Experimental Results

The Prime Miner is part of the *nightly build* of ProM Tools at http://www.promtools.org/doku.php?id=nightly. We encourage the reader to download the ProM *nightly build* and try the Prime Miner. To discuss the advantages and disadvantages of our presented approach, we apply the miner to two more well-known examples in this section. The examples are the *reviewing example* and the *teleclaims example* introduced in Chapter 8 of the book *Process Mining: Data Science in Action* by Wil van der Aalst [1]. The respective files can be found at http://www.processmining.org/event_logs_and_models_used_in_book.

The *reviewing example* is an event log describing the process of handling reviews for a journal. The event log records $100$ cases and $3730$ events. The control flow of the process is quite simple, but includes a large number of choices, which leads to a huge set of different runs. Using no concurrency oracle, the Prime Miner will generate $96$ different Hasse diagrams. We can see that it is an artificial event log after all, built to cover all different choices. Using the $\alpha$-oracle, the Prime Miner will detect $12$ pairs of concurrent actions, resulting in no less than $93$ different Hasse diagrams. Using the implementation of the Prime Miner in ProM, we can see all of the $93$ diagrams. Every diagram starts with the invitation of reviewers, then there are three concurrent events of three reviewers submitting or not submitting reviews. This first part of every run is then followed by a long sequence of events modeling the looping behavior of the rest of the process. The main problem here is that there is almost no reduction when going from the language to the partial language. The partial language is far too big to be handled by any regular region-based approach. Using the Prime Miner, we can choose to only consider the five most frequent runs of the event log to obtain $4$ Petri net models in $3$ minutes. Net $1$, synthesized from Hasse diagram $1$, can execute $5.0\%$ of the traces. This net is the most frequent Hasse diagram, where reviewer $2$ does not submit and the paper is rejected immediately. Net $2$, synthesized from the prime event structure built from Hasse diagrams $1$ and $2$, can execute $10.0\%$ of the traces. This net is quite complex. It shows the alternatives of reviewer $2$ submitting, additional reviews, and accepting the paper. Net $3$, synthesized from the prime event structure built from

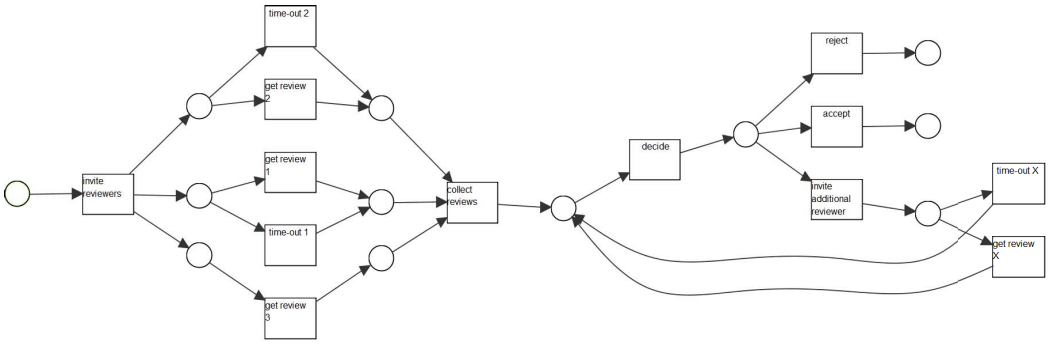



Figure 5. Petri net modelling the reviewing example.

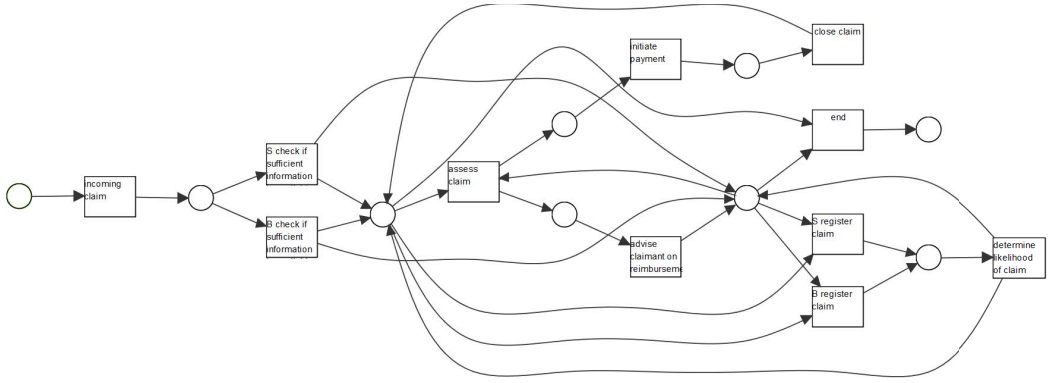



Figure 6. Petri net modelling the teleclaims example.

Hasse diagrams 1, 2, and 3, can execute 22.0% of the traces. Net 4, synthesized from the prime event structure built from Hasse diagrams 1, 2, 3, 4, and 5, can execute 50.0% of the traces. Net 3 and Net 4 (Net 4 is depicted in Figure 5) are very reasonable models of the *reviewing example*. Net 3 shows the complete process together with the choice of reviewer 2 submitting or not. Net 4 shows the same process but adds the choice for reviewer 1. Considering more Hasse diagrams, the region theory approach is too slow to obtain a result in reasonable time. However, the results generated by building the prime event structure from only 5 of the 93 different runs are surprisingly useful. This is how we use the Prime Miner on big event logs to obtain reliable and precise models for the most frequent runs.

The *teleclaims example* is an event log recording the handling of claims in an insurance company. The log contains 46138 events related to 3512 cases. The process deals with the handling of phone calls where different types of insurance claims are logged. The control flow of the process is quite simple yet, the event log is difficult to handle for most mining algorithms [1]. The reason is that cases are ended at different stages of the process using an explicit *end* action. This is problematic because ignoring the *end* actions, the process is actually driven by only two runs. Now, however, there are many cases that are prefixes of the complete process but they have an additional *end* event so that they need to be considered as independent and complete cases. If we just delete all the end events, every mining algorithm will create a good model in no time. If an algorithm supports label splitting, mining will result in five different *end* events, but label splitting is counterintuitive when building an integrated model. We can delete all end events in a pre-processing phase before starting the Prime Miner to obtain a perfect model out of 2 runs in almost no time. This model is just as good as the model from using extensive label splitting. If we want to model the unique *end* action explicitly and as recorded, we apply the Prime Miner to the original event log without using a concurrency oracle. We then obtain 12 Hasse diagrams and 6 Petri net models in 2 seconds. Net 6 (depicted in Figure 6), synthesized from the prime event structure built from Hasse diagrams 1, 2, 3, 4, 6, 7, 8, 9, 10, and 12, can execute 86.0% of all traces. The net looks quite complex but it is a sound workflow net with unique labels explicitly modeling the *end* action. As we know from region theory approaches, this net is a quite precise result. The reason why the Prime Miner is able to calculate this net is described above in Section 2 (f). At some point during the mining procedure, the Prime Miner builds a prime event structure from the first 5 Hasse diagrams and synthesizes a Petri net. This net is not connected, i.e., it has transitions with an empty prefix. As stated in Section 2 (f), diagram 5 is then discarded. The same happens to Hasse diagram 11. In that sense, the Prime Miner will refuse to integrate non-fitting runs and will output only connected

workflow models. We use minimal regions to obtain places that are empty at the end of every run if possible. Summing up, the Prime Miner tends to generate sound Petri nets, even in examples with complex control flow.

## 4. Conclusion

The Prime Miner generates fitting and readable results regarding the most frequent runs of an event log. The obvious limitation of the Prime Miner is run-time. The run-time of every region-based approach will grow exponentially with the size of the input. Using tokenflow regions, the set of all regions might grow exponentially with the number of arcs of the prime event structure. We only calculate a minimal set of these regions but this is still very costly. Furthermore, recalculating the synthesis result over and over again adding more runs does not help the run-time issues. To tackle the run-time, we synthesize one-safe Petri nets and regard the most frequent runs only.

Summing up, we presented a new approach to processing discovery based on regions for prime event structures. To deal with the common problems of region-based algorithms, we applied an iterative approach based on frequencies of runs. We obtained a new discovery method, tackling the problem of discovery from a new perspective: The Prime Miner is able to produce controllable and precise synthesis results for a set of frequent use-cases of a business process. We implemented the Prime Miner for ProM.

In this paper, we discussed the properties of the miner on well-known standard data-sets. The next step is to apply the approach to real-life data-sets and compare the results to other techniques regarding standard quality measures. Furthermore, we would like to take the presented ideas even further. We could choose any set of runs to be folded into a prime event structure. As an example, think of the Hasse diagrams 5 and 11 of the *teleclaims example*. We would like to implement more and different concurrency oracles, maybe even consider concurrency on the level of events. Furthermore, we would like to investigate the influences of different, more general region definitions. Last but not least, we have to tune the run-time even more if we try to tackle bigger sets of runs in real-life data sets.

## References


[1] van der Aalst, W. M. P.: *Process Mining: Data Science in Action*. Springer, 2016.

[2] van der Aalst, W. M. P.; van Dongen, B. F.: *Discovering Petri Nets from Event Logs*. ToPNoC VII, LNCS 7480, Springer, 2013, 372–422.

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

[4] van der Aalst, W. M. P.; Weijters, T.; Maruster, L.: *Workflow Mining: Discovering Process Models from Event Logs*. IEEE Trans. Knowl. Data Eng. 16(9), 2004, 1128–1142.

[5] Armas Cervantes, A.; Dumas, M.; La Rosa, M.; Maaradji, A.: *Local Concurrency Detection in Business Process Event Logs*. ACM Transactions on Internet Technology 19 (1), 2018.

[6] van Beest, N.; Dumas, M.; García-Bañuelos, L.; La Rosa, M.: *Log Delta Analysis: Interpretable Differencing of Business Process Event Logs*. BPM 2015, LNCS 9253, Springer 2015, 386–405.

[7] Bergenthum, R.: *Synthesizing Petri Nets from Hasse Diagrams*. BPM 2017, LNCS 10445, Springer, 2017, 22–39.

[8] Bergenthum, R.; Meis, B.: *Mining with Eve - Process Discovery and Event Structures*. Workshop ATAED 2017, CEUR 1847, 2017, 71–75.

[9] Bergenthum, R.; Lorenz, R.: *Verification of Scenarios in Petri Nets Using Compact Tokenflows*. Fundamenta Informaticae 137, IOS Press, 2015, 117–142.

[10] Bergenthum, R.: *Faster Verification of Partially Ordered Runs in Petri Nets Using Compact Tokenflows*. Petri Nets 2013, LNCS 7927, Springer, 2013, 330–348.

[11] Bergenthum, R.; Mauser, S.: *Folding partially ordered runs*. Workshop ART 2011, CEUR 725, 2011, 52–62.

[12] Bergenthum, R.; Desel, J.; Mauser, S.: *Comparison of Different Algorithms to Synthesize a Petri Net from a Partial Language*. ToPNoC III, LNCS 5800, Springer, 2009, 216–243.

[13] Bergenthum, R.; Desel, J.; Lorenz, R.; Mauser, S.: *Process Mining Based on Regions of Languages*. Business Process Management 2007, LNCS 4714, Springer, 2007, 375–383.

[14] Carmona, J.; Cortadella, J.; Kishinevsky, M.: *New Region-Based Algorithms for Deriving Bounded Petri Nets*. IEEE Trans. Computers 59(3), 2010, 371–384.

[15] Dumas, M.; García-Bañuelos, L.: *Process Mining Reloaded: Event Structures as a Unified Representation of Process Models and Event Logs*. Petri Nets 2015, LNCS 9115, Springer, 2015, 33–48.

[16] van Dongen, B.; Alves de Medeiros, A. K.; Verbeek, H. M. W.; Weijters, A. J. M. M.; van der Aalst, W. M. P.: *The ProM framework: A New Era in Process Mining Tool Support*. Petri Nets 2005, LNCS 3536, Springer, 2015, 444–454.

[17] van Dongen, B.; van der Aalst, W. M. P.: *Multi-Phase Process Mining: Aggregating Instance Graphs into EPCs and Petri Nets*. Applications of Petri Nets to Coordination, Workflow and Business Process Management 2005, 2015, 35–58.

[18] van Dongen, B.; Desel, J.; van der Aalst, W. M. P.: *Aggregating Causal Runs into Workflow Nets*. ToPNNoC VI, LNCS 7400, Springer, 2012, 334–363.

[19] Fahland, D.; van der Aalst, W. M. P.: *Simplifying discovered process models in a controlled manner*. Inf. Syst. 38(4), 2013, 585–605.

[20] IEEE Task Force on Process Mining: *Process Mining Manifesto*. Business Process Management Workshops, LNBIP 99, Springer, 2012, 169–194.

[21] Leemans, J. J. S.; Fahland, D.; van der Aalst, W. M. P.: *Discovering Block-Structured Process Models from Event Logs - A Constructive Approach*. Petri Nets 2013, LNCS 7927, Springer, 2013, 311–329.

[22] Lu, X.; Fahland, D.; van der Aalst, W. M. P.: *Conformance Checking Based on Partially Ordered Event Data*. Business Process Management Workshops 2014, LNBIP 202, Springer, 2014, 75–88.

[23] Nielsen, M.; Plotkin, G.; Winskel, G.: *Petri Nets, Event Structures and Domains, Part I*. Theoretical Computer Science 13, Elsevier, 1981, 85-108.

[24] Ponce de León, H.; Rodríguez, C.; Carmona, J.; Heljanko, K.; Haar, S.: *Unfolding-Based Process Discovery*. Automated Technology for Verification and Analysis 2015, LNCS 9364, Springer, 2015, 31–47.

[25] Reisig, W.: *Understanding Petri Nets - Modeling Techniques, Analysis Methods, Case Studies*. Springer, 2013.

[26] Weijters, A. J. M. M.; Ribeiro, J. T. S.: *Flexible Heuristics Miner (FHM)*. CIDM, IEEE 2011, 310-317.

[27] van der Werf, J.; van Dongen, B.; Hurkens, C.; Serebrenik, A.: *Process Discovery Using Integer Linear Programming*. Petri Nets 2008, LNCS 5062, Springer, 2008, 368–387.

# Process mining of logged gaming behavior

Souad Ramadan, Halim Ibrahim Baqapuri, Erik Roecher, Klaus Mathiak
Department of Psychiatry, Psychotherapy and Psychosomatics
RWTH Aachen University
Aachen, Germany
e-mail: sramadan@ukaachen.de

***Abstract*—Video gaming takes a significant position in contemporary social life. It also constitutes as an elegant scientific model to study social behavior. By applying this model, behavioral patterns and their underlying physiological processes can be precisely observed at millisecond resolutions, e.g. with functional brain imaging. To date, a detailed behavioral analysis on an event-level was only performed in a few studies. Process mining (PM) is an efficient methodology to assess and systematize these complex behavioral patterns.**

**We describe the application of PM to semi-naturalistic behavior during gameplay of a first-person shooter (FPS). Behavioral data were collected from 18 participants performing three 10-min gaming sessions of a customized FPS during functional brain imaging. An event log recorded during gameplay represented 10 types of game events in chronological order. In total 54 sessions were entered in the analysis serving as process indicators (cases). ProM software (version 6.8 TU Eindhoven, NL) provided data summary, heuristic and fuzzy PM.**

**Event summaries revealed that being under attack early on prevented environmental exploration and resulted in shorter behavioral cycles, in contrast to other behaviors such as navigation and active attacking. Furthermore, the Fuzzy Miner generated a general overview of the player behavior by defining gaming pattern sequences. Upon further investigation, the Heuristic Miner revealed examples of how PM could highlight important patterns in player behavior. These patterns reflect the uninfluenced behavior ('free will') of the player and the player's tendency towards aggressive or defensive behavior.**

**PM was used to analyze and systematize behavioral patterns in complex semi-naturalistic behavior during game play and showed capability to give an overview of this behavior and then further describe detailed information about them. This can allow for more in-depth investigations of interactive social behavior in the future. Furthermore, the capability of inspecting behavioral cycle characteristics (length, performance) in varied situations enables the use of PM as a testing tool for the preparation of game related paradigms in functional brain imaging. Finally, the description of such patterns and their alterations in psychiatric disorders may be applied to investigate therapeutic targets for social impairments.**

***Keywords-component; Process mining, fMRI, Behavioral patterns, First-person shooter, ProM, Semi-naturalistic behavior***

## I. INTRODUCTION

video games play a significant role in the contemporary entertainment industry. According to the Entertainment Software Association, 60% of Americans play video games daily. Shooter games represent 25.9% of the best-selling video game genres of 2017 [1]. Moreover, the popularity of video games goes beyond the entertainment world, and has been considered as an elegant model to study complex social behavior [2]. Such a model enables precise observation of behavioral patterns and the analysis of their underlying physiological processes, even at a millisecond (ms)-resolutions, e.g. with functional brain imaging.

### *A. Video games:*

A review paper described a significant increase in scientific studies on the neural and behavioral effects of video games [3]. Since 2005, published studies about video games or studies that use a form of gaming have been increasing 20% per year and within these studies 57.8% of them used gaming as the experimental manipulation [3]. Video gaming allows the rehearsal of realistic situations without the risk involved and can be considered as a rehearsal of semi-naturalistic behaviors as well [4]. This makes it possible to observe and study semi-naturalistic behaviors for conditions that aren't easy to replicate and test in real life, such as aggressive behaviors. First-person-shooter (FPS) games have been shown to display realistic virtual violence [5] and according to the literature, FPS games are one of the main gaming genres which are designed to engage the player individually or with other characters in violent virtual activities [6].

However, despite the potential of video games as research environments, only a few studies have established systematic content analysis on event-level in video games, especially FPS games [2][7]. Studies on video games that analyze violent virtual content, generally use indicators of typical content such as the percent of physical aggression [8] or the rate of violence in "E"-rated console games [9], and do not delve into detailed event level analysis. The few studies that do implement event-level analysis of FPS games, generally depended on frame by frame visual classification of recorded videos of the gameplay. Such procedure is limited by its requirement of human time

978-1-7281-0919-0/19/$31.00 ©2019 IEEE
DOI 10.1109/ICPM.2019.00019

resources, interrater reliability, and may introduce biases due to visual observation of events to define phases[7].

### B. *Process mining:*

Process mining (PM), also known as workflow mining [10], is a technology that infers a process from an event or activity log and can also display this process as a graphical output. Furthermore, it supports subsequent analysis by creating process models, e.g., heuristic or fuzzy approximations. Visual representations in general are easy to understand and thus provide a good basis to investigate complex semi-naturalistic behavioral patterns, e.g., in FPS games. The obtained models depend on actual execution of the processes, providing insight about underlying mechanisms or factors influencing the process structure [11] [12].

The open source software package ProM (version 6.8, Eindhoven University E/TU, NL) was selected to conduct the data analysis for the current study. ProM offers several PM algorithms for the analysis of event data and can support the extraction of specific and repetitive behavioral patterns [13]. The event log is the main component inside PM, it is a set of events where each single event occurs at a given point in time and there is an interval time between each event [14].

ProM implements several plug-ins, which are divided according to their function into three categories:

• Discovery: These plug-ins are purely based on the event log data.

• Conformance: These plug-ins check how the data in the event log matches with the prescribed behavior in the deployed model.

• Extension: These plug-ins use both a model and the event logs to discover information that will enhance the said model [15].

A set of Discovery algorithms was designed to reduce the complexity of the model as much as possible such as in heuristic mining [16], Fuzzy-logic [17] or genetic process mining [18]. For our study, the Fuzzy Miner and the Heuristic Miner were used to simplify the complex nature of the behavioral data as recorded by the event log.

This is an explorative case study, investigating whether PM could prove to be a sufficient analytical tool for studying behavioral patterns in complex semi-naturalistic settings. We hypothesize that PM will be able to successfully generate processes that represent behavioral cycles during gameplay. Lastly, we also intend to investigate if PM can be proposed as an effective tool to produce behavioral processes for correlational analysis with fMRI brain data in the future.

## II. METHODOLOGY

The study was conducted using MATLAB R2016b (The MathWorks, Inc., Natick, MA), a spreadsheet program (Excel 2016), and ProM software (version 6.8, TU/Eindhoven University, NL). The work-flow comprised several steps outlined in the following subsections and shown in Fig. 1.

### A. *Data collection:*

Behavioral data were collected from a specially modified version of a FPS game inside an MRI scanner during functional brain imaging. The FPS game was modified from the open source project 'Shooter Game' using Unreal Engine (version 4.12), available online from Epic Games Inc. The game was designed as an arena FPS where two or more opposing teams or individuals contest to get the highest amount of points (score) within a certain time limit. The players in the FPS game are also encouraged to navigate the arena to find and acquire pick-ups which help them score in the arena.

We included 18 participants for this study. During our paradigm each participant was always competing against one artificial intelligence (AI) controlled 'Bot' character for the highest score. Participants got scored based on either successfully eliminating the AI Bot or successfully avoiding being eliminated from the AI Bot. Each participant played three sessions of the game within the scanner and each session was 10 minutes long.

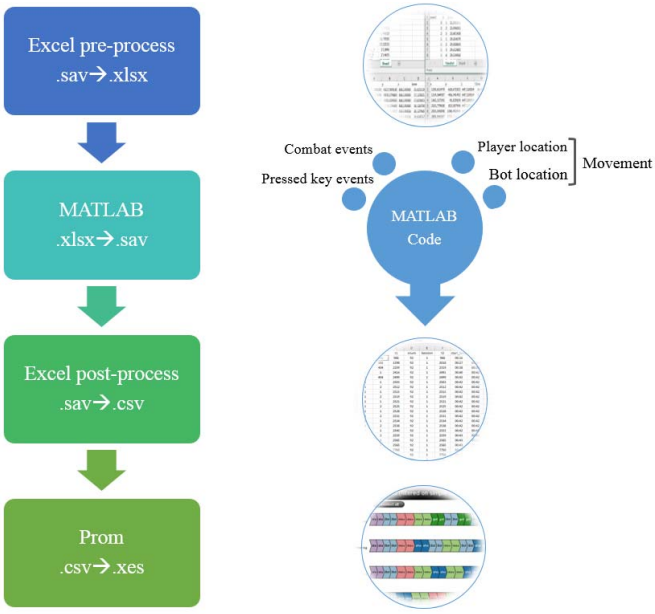



Figure 1. A flowchart of the main study steps combined with the data evolution on the right side. After pre-processing with sorting and removal of doublets, movement parameter were extraction and format conversion for ProM import applied.

### B. *Data preparation:*

After each session the game generated a log file with timestamps which detailed the actions of the participant within the session. This data was then imported into a spreadsheet program (Excel). There were minor inconsistencies of starting and ending time between the log

files of different sessions introduced due to manual starting of each gaming session with the fMRI sequence in the scanner. These inconsistencies were removed by filtering the data in Excel to strictly defined session start and end times (from 00:00:14 until 00:10:30). This data was then reformatted into purely numeric data for easy export to MATLAB.

### C. *Event log construction:*

Our event log was assembled from 4 event files x 3 sessions x 18 participants. MATLAB was used to:

1. Define the player movement direction relative to the Bot location by comparing the location of the player and the Bot simultaneously.
2. Collect the coded events from the four event files and set them in chronological order. There are 10 separate events stored in these files and they are defined as follows:

- '1' Bot is shooting.
- '2' Player is shooting.
- '3' Player's shot hits the Bot.
- '4' Bot's shot hits the player.
- '5' Player dies.
- '6' Player kills the Bot.
- '7' Health package is picked up.
- '300' Beginning of the round.
- '404' Moving towards the Bot.
- '505' Moving away from the Bot.

3. Eliminate event repetition by assigning each event, a duration. Hence, each event was labeled with a 'Start' or a 'Complete' tag. Here it is important to note that the events '1', '2', '300', '404', and '505' are continuous events, meaning that they occur over a period of time. Whereas the events '3', '4', '5', '6', and '7' are instantaneous events, meaning that they start and end at the exact same time point. This distinction is made within ProM at the 'Event log pre-preprocessing' step described later in the paper.
4. Define 'Behavioral cycle' (round), which began with event '300' and ended with event '5'. Each session yielded several behavioral cycles.

The resulting events were then imported back into Excel and converted into a comma separated format to be passed to the process mining tool. The event log contained synchronized and sequential events in a table format consisting of eight main columns:

i. Round: Referring to the behavioral cycle.
ii. Event.
iii. Snum: Referring to the subject number. It is used to combine each of the 3 sessions together for each participant.
iv. Session: Referring to the gaming session.

TABLE I. EXAMPLE FOR EVENT LOG

| Roun -d | Even -t | t1 | S num | Sessio n | t2 | Start time | End time |
|---|---|---|---|---|---|---|---|
| 1 | 300 | 966 | 92 | 1 | 966 | 00:16 | 00:16 |
| 1 | 505 | 1598 | 92 | 1 | 2018 | 00:27 | 00:34 |
| 1 | 404 | 2259 | 92 | 1 | 2319 | 00:38 | 00:39 |
| 1 | 1 | 2416 | 92 | 1 | 2491 | 00:40 | 00:42 |
| : | : | : | : | : | : | : | : |
| : | : | : | : | : | : | : | : |
| 1 | 1 | 7743 | 92 | 1 | 7750 | 02:09 | 02:09 |
| 1 | 404 | 7756 | 92 | 1 | 7756 | 02:09 | 02:09 |
| 1 | 1 | 7762 | 92 | 1 | 7762 | 02:09 | 02:09 |
| 1 | 2 | 7770 | 92 | 1 | 7770 | 02:09 | 02:09 |
| 1 | 1 | 7775 | 92 | 1 | 7775 | 02:10 | 02:10 |
| 1 | 4 | 7775 | 92 | 1 | 7775 | 02:10 | 02:10 |
| 1 | 2 | 7776 | 92 | 1 | 7776 | 02:10 | 02:10 |
| 1 | 1 | 7781 | 92 | 1 | 7781 | 02:10 | 02:10 |
| 1 | 4 | 7781 | 92 | 1 | 7781 | 02:10 | 02:10 |
| 1 | 5 | 7781 | 92 | 1 | 7781 | 02:10 | 02:10 |
| 2 | 300 | 7782 | 92 | 1 | 7782 | 02:10 | 02:10 |
| 2 | 1 | 7787 | 92 | 1 | 7793 | 02:10 | 02:10 |
| 2 | 505 | 8086 | 92 | 1 | 8207 | 02:15 | 02:17 |
| : | : | : | : | : | : | : | : |

Example for event log components, first column refers to the behavioral cycle which is defined as the period from event '300' 'Start' until event '5' 'Death', and we can see the beginning of the second behavioral cycle in row 18. The first behavioral cycle was trimmed in the middle to give a general overview, the red dashed rectangle highlights synchronized events.

v. t1 and t2: Event start and end time in frames (2 columns).
vi. Start_time and End_time: Event start and end time converted into 'minutes:seconds' format (2 columns).

Table I shows an example of the event log components. Each row corresponds to one event, with the dashed red boxes highlighting an example of synchronized events.

### D. *Process mining:*

The process mining toolbox ProM (6.8, TU/Eindhoven, NL) was chosen to study and analyze the semi-naturalistic behavior during gameplay. Using the MATLAB coding scheme, an event log was created with the session number defined as a Case ID and the Start_time and End_time defined as a timestamp. With this we obtained 54 cases (Traces) with lifecycle transitions for each event (Start – Complete).

#### 1) *Event log summary:*

The ProM software provided a log summary for the event log which indicated that event '1' 'Bot shooting' represents 33.97% of all events while event '2' 'Player shooting' represents 29.33% of all events. The events '404' and '505'occur 12.11% and 5.80% of the time respectively, which reflects the movement of the player as shown in Table II.

TABLE II EVENT TYPES

| Class | Event name | Occurrences (absolute) | Occurrences (relative) |
|---|---|---|---|
| 1 | Bot shooting | 34374 | 33.96% |
| 2 | Shooting | 29680 | 29.33% |
| 404 | Moving toward bot | 12258 | 12.11% |
| 4 | Got shot | 8146 | 8.05% |
| 505 | Moving away from bot | 5874 | 5.80% |
| 3 | Hit | 5774 | 5.71% |
| 6 | Kill | 1508 | 1.49% |
| 300 | Start | 1336 | 1.32% |
| 5 | Death | 1228 | 1.21% |
| 7 | Pick up health | 1022 | 1.01% |

The table represents the absolute and relative occurrences of the Classes (events) provided by the ProM software. The column Event name was added to explain the event coding in the first column.

ProM was also used to create a dotted chart of the event log and the chart was color coded according to the round. This gave us an overview of each round's duration in the event log, see Fig. 2. Here it was easy to spot the high variance in the duration of the first rounds between each case.

*2) Event log pre-processing:*

We applied the 'Simple Heuristics' filter model from ProM to remove the lifecycle transition (Start – Complete) for instantaneous events. They were represented in the event log only as a complete event. Continuous events kept their lifecycle transition as described earlier and this new event log was named filtered event log 1 (fEL-1). We were interested in the behavior of the player and as such wanted to focus on events that gave more information about the player. Since event '1' described mostly the behavior of the Bot, it was removed from the event log.

Corresponding events '2', '3', and '4' gave us relevant information from event '1' while still focusing on the behavior of the player. This new event log was called filtered event log 2 (fEL-2).

*3) Miner application:*

*a) Simple Heuristic filter:*

As noted earlier, there was high variance in the duration of the first rounds between the cases. We wanted to investigate if this had any effect on the player behavior or performance. Therefore, we separated the 54 obtained cases in fEL-1 using the following filtering steps: Firstly, we filtered the event '300' out of fEL-1 using the 'Event Attribute Values' filter in ProM.

This was done so that the cases could be grouped using different first events by ProM. Secondly, the Simple Heuristics filter was used to only keep cases that began with direct sudden combat (DSC), which was either event '1' or '2'. This was done as we suspected that the variance in the first round durations was driven by early combat. Hence, we obtained two new event logs with 27 cases each. The first event log, DSC, only had cases that began with either event '1' or '2' and the second event log, indirect combat (IC), had cases that began with any other event. We generated dotted charts for both of these event logs and they are shown in Fig. 3 and Fig. 4 respectively. It can be seen in these figures that the separation has removed the high variability of the first round durations within the DSC and IC event logs.

To be able to compare the difference in performance between the DSC and IC event logs, we calculated a ratio of killing (RoK). This was done by adding the amount of kills in each first round of a case and then dividing this by the duration of the said first round to account for differences in the durations. This gave us a ratio of kills per first round, regardless of the round duration. To compare the overall performance of the DSC and IC logs, we calculated the average number of kills, the average number of deaths, and the average number of health packages collected.

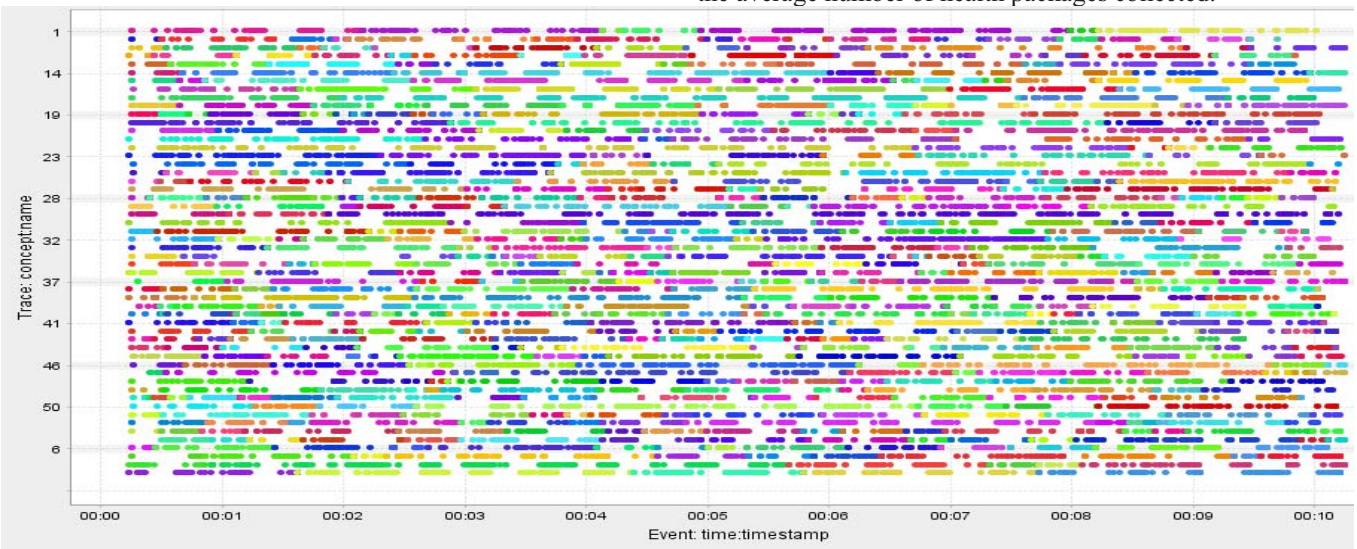



Figure 2. Dotted chart of event log with 54 cases, colored by behavior cycle. e.g.in the first Trace (case) there are 6 behavior cycles, the first one lasts for around 2 minutes. Event time is given in minutes.

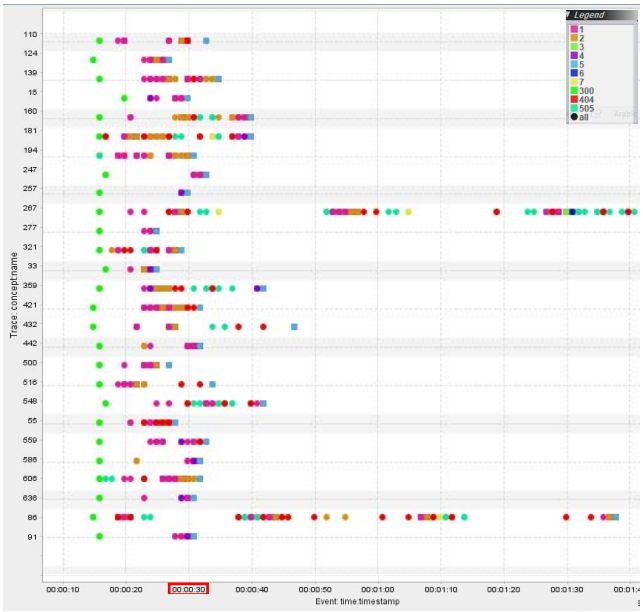



Figure 3. Dotted chart of event log with 27 cases first behavioral cycle that start with direct sudden compact (DSC), colored by event name, red rectangle on the 30 second mark, legend at the top right corner.


Figure 4. Dotted chart of event log with 27 cases first behavioral cycle that start with indirect combat (IC), colored by event name, red rectangle on the 30 second mark, legend at the top right corner.

*b) Fuzzy Miner*

We used the Fuzzy Miner in ProM to look into the main behavioral patterns in fEL-2. The Fuzzy Miner has a feature that animates data, giving insight into the generated model. We were able to use this feature to identify which events were considered as main actions performed by the player. This guided our subsequent analysis with the Heuristics Miner. Fig.5 shows a general model representing the playing behavior using the fEL-2 event log.

*c) Heuristics Miner*

We used the 'Interactive Data-aware Heuristics Miner' within ProM to look into behavioral patterns with more detail. The Interactive Data-aware Heuristics Miner was used to generate 'Directly-Follow-Graphs'. These graphs represent the frequency with which an event follows the immediate neighbor in the event log, which in our case was fEL-2.

The Directly-Follow-Graphs visualized the player behavior and allowed us to highlight the paths that we were interested in while the Heuristic miner provided the numeric statistics for these behaviors.

## III. RESULTS

### *A. Simple Heuristic filter:*

The separation of fEL-1 based on first round duration created the DSC and IC event logs. Out of the 18 participants, only 4 participants' first rounds (in all 3 sessions) were exclusive to either the DSC or IC event log. The rest of the 14 participants had their first rounds (in all 3 sessions) present in both the DSC and the IC event log. The comparison between the DSC and IC event logs showed that the average duration of the first behavioral cycle in DSC (24.67 s) was significantly shorter than in IC (77.4 s) ($t(52) = 4.9$, $p<0.0001$), as shown in Fig. 6. The average duration of all behavioral cycles was 47.13 s. While comparing the overall performance between DSC and IC event logs, results show significantly higher number of average kills ($t(52) = 4.9$, $p <0.04$) (see Fig. 7) in the IC cases compared to the DSC cases, a significantly lower number of deaths in the IC cases compared to the DSC cases and a significantly higher occurrence of picking up health packages in IC cases compared to the DSC cases.

Comparing the RoK, to account for varying first round durations, we saw a significantly higher RoK ($t (52) = 2,75$, $p=0.005$) of the IC cases as compared to the DSC cases (see table III).

### *B. Fuzzy Miner*

An overview of the participant's semi-naturalistic behavior was obtained by the Fuzzy Miner with a confidence of 89.24% on fEl-2 log, as shown in Fig. 5. The Fuzzy Miner discarded less significant events (below significant cutoff; for our study it was set to zero) and lowly correlated events from the process model. The significance of an event is shown as a number inside the event boxes. This number can range from 0 to 1, where 1 is the most significant event and 0 the least. The arrows show the process direction and their thickness and darkness show the frequency of the events. The model shows the behavior as a loop that begins with the start event '300' and ends generally by the death event '5'. When we removed event '1' to make fEL-2, some self-repetition loops on different events occurred in this model. These were represented in the model as looped arrows over some events and were manually removed for easier model visualization.

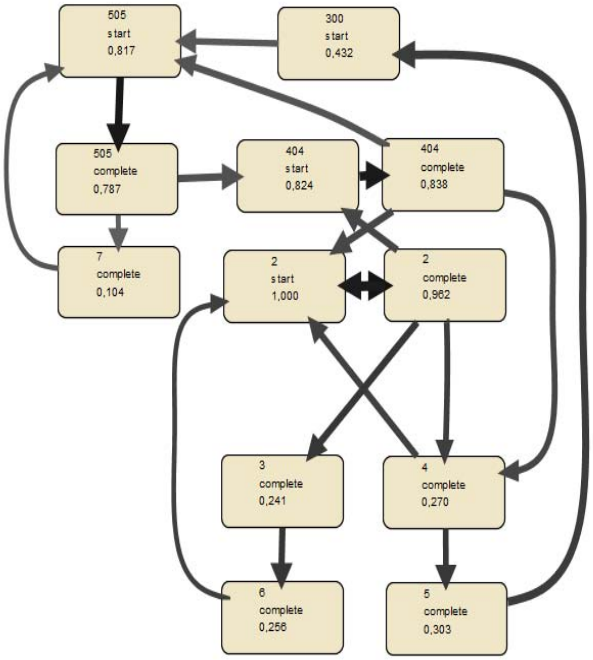



Figure 5. Fuzzy model of semi-naturalistic behavior during the FPS game play after removing event '1'. Each box has the event class, lifecycle transition (Start – Complete), and a number that indicates its significance (maximal value is 1.0), Only continuous event kept the lifecycle transition (Start – Complete). Legend: '2': Shooting, '404': Moving toward bot, event '5': Death, '4': Got shot, '505': Moving away from bot,'3': Hit, '6': Kill,'7':Pick up health.

### C. *Heuristics Miner*

The overall behavior as represented by the Directly-Follow-Graph is shown in Fig. 8. This was generated using the fEL-2 with a fitness factor of 0.99. Fig. 9 and Fig. 10 show two examples of the usefulness of this approach. The Directly-Follow-Graph allows us to choose any event and see how other events are related and connected to it. Fig 9 shows all that the events connected to event '4'. The main outputs of the event '4' (Bot's shot hitting the player) were event '2' (Player shooting) and '404'(Moving towards Bot) with 47.75% ,17.0% occurrence. This was in contrast to the 4.77% occurrence for the event '505'(Moving away from Bot) following event '4'.

Fig. 10 shows that for event '300' the main output is event '505' (Moving away from Bot) with an occurrence of 65.86% as compared to the event '404' (Moving towards Bot) with only an occurrence of 13.66%.

TABLE III RATIO OF KILLING

| Average of: | IC | DSC | Notice |
|---|---|---|---|
| RoK | 0,0201 | 0,0097 | t (52) = 2,75, p=0.005 |

The table represents comparison of the ratio of killing with values of t-test in the Notice. IC: Cases with first behavioral cycle with indirect combat. DSC: Cases with first behavioral cycle with direct sudden. RoK: ratio of killing.

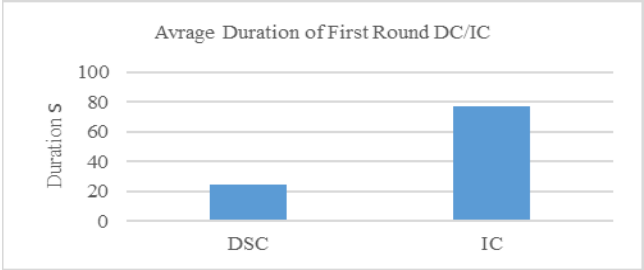



Figure 6. Comparison between the average duration of direct compact for the first round (DSC) and indirect combat (IC).

## IV. DISCUSSION AND CONCLUSION

In this study, we have applied PM techniques to the semi-naturalistic behavior during gameplay of our FPS game. Our customized FPS game has a very limited number of event types which limits the behavioral patterns that can be generated by PM models.

However, even with this limited range of events we could use PM to show its application in this field. Firstly, the behavioral models obtained by the Fuzzy Miner and its animation-based exploration confirmed that shooting is the most common behavior pattern followed by navigation in the arena space. Secondly, the comparison between event logs, revealed that the behavioral cycles were largely affected by initial interaction. Being directly under sudden attack, shortened the behavioral cycles and hindered the overall performance during the case as well as hindered the performance during the current cycle, as shown by a lower RoK. In contrast to this, having the opportunity to explore the environment early on led to better performance (see Fig. 6 and 7; table III and IV). This pattern may be a target for neuropsychological research; whether it reflects behaviors alike 'learned helplessness' where experience of insufficiency leads to passive behavior [19].

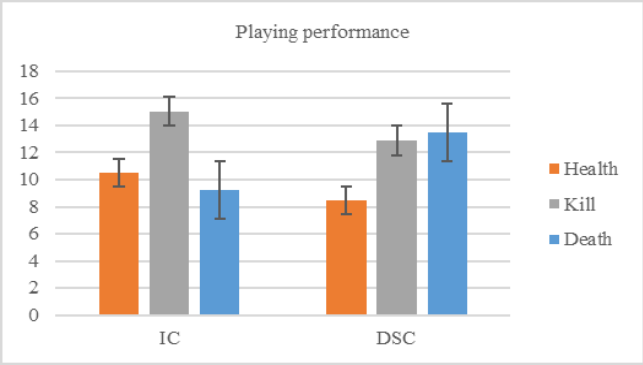



Figure 7. Playing performance comparison between cases that start with indirect sudden combat IC and direct combat DSC.

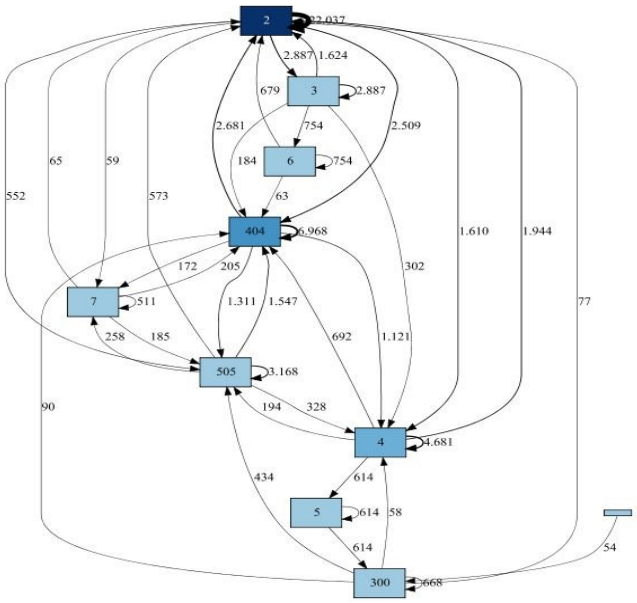



Figure 8. Directly-Follow-Graph shows the general graph of the participant behavior without event '1': Bot shooting. The arrows refer to the process direction, the box color to frequency and has the event number on it. Legend: '2': Shooting, '404': Moving toward bot, event '5': Death, '4': Got shot, '505': Moving away from bot,'3': Hit, '6': Kill,'7':Pick up health.

Investigating the behavioral patterns, two examples emerged which highlight the usefulness of PM in the analysis of semi-naturalistic behavior during video gaming. The Heuristic Miner exposed a spatial exploration pattern occupying a notable position in the participant's behavior, leading them to enter combat patterns later. Fig. 10 showed the high frequency occurrence of the retreat pattern (event '505' 'Move away from bot') directly after the

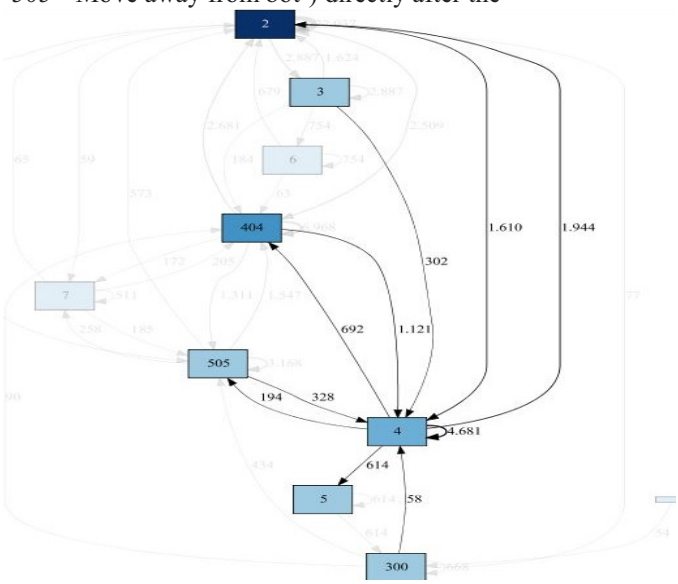



Figure 9. Directly-Follow-Graph shows highlighted connection (input, output) of event '4':' Got shot'. Input event: '4', '2','404','505','300','3'. Output event: '2', '404', '5', '4', '505' '7', Legend: '2': Shooting, '404': Moving toward bot, event '5': Death, '4': Got shot, '505': Moving away from bot,'3': Hit.

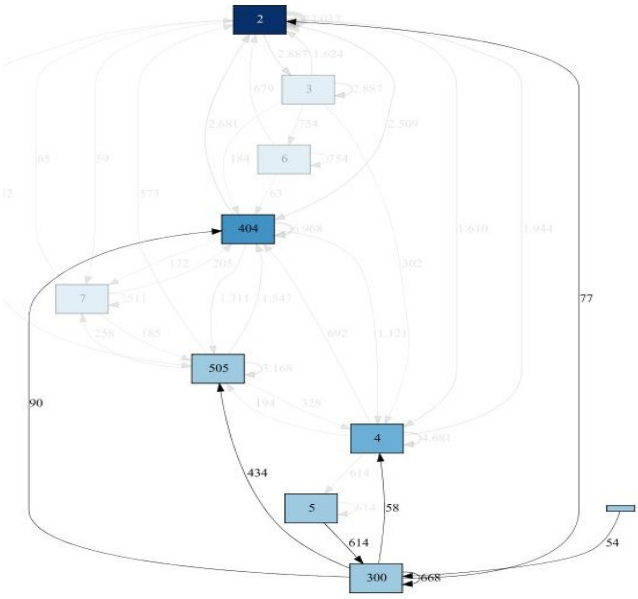



Figure. 10. Directly- Follow-Graph shows highlighted connection (input, output) of event '300': 'Start'. Input event: '5', '300'. Output event: '505', '404' '2', '4'. Legend: '505': Moving away from bot, '404': Moving toward bot, '2': Shooting '4': Got shot, '300': Start.

beginning of the behavioral cycle, i.e., the event '300' (Start of round). This differential activation and retreat pattern may be seen in the framework of the behavioral activation-inhibition system and can be investigated in future neurophysiological analyses [20]. Even though movement is a relative event in our case (as the direction of movement was defined relative to the bot location), there are still different patterns that can be shown based on this. The retreating pattern from the player shows a tendency to explore the environment before any direct combat. Whereas, the advancing pattern that the players exhibited, showed a higher tendency to engage into further combat after being hit versus not being hit. This was confirmed by the statistics in the Heuristic miner and supported by the Directly-Follow-Graph as shown in the outputs of event '4' (see Fig.8). Future neurophysiological analyses could investigate whether this modulation of aggressive behavior is linked to the General Aggression Model and may help to explain learning of aggressive responses in virtual and real social interaction [21].

We wanted to test if PM could be used as an analytical tool to extract semi-naturalistic behavior from video game data. We have shown in this case study that given sufficient, and in some cases limited, event data, PM can indeed generate processes which reflect player behavior. The event logs which are created as a part of PM can be used, with some pre-processing, as regressors for correlational fMRI data analysis. We can also conclude that the models that are generated from these event logs can help us make more informed conclusions about behavioral implications of fMRI data.

In comparison to previous video game event-level analysis techniques, PM can generate data driven semi-naturalistic behavioral models from gaming events. These models offer insightful overviews into behavioral patterns

and PM tends to be very functional in letting users visualize and animate complex behavior in simple illustrations. However, a major limitation of this approach is that simultaneous events are not well represented in the visualizations. The Directly-Follow-Graphs visualize all events in the event log sequentially, meaning that concurrent events are not represented well in them and are instead viewed as subsequent events. With respect to neuropsychological and fMRI analysis, a focus needs to be taken on finding models that create large enough number of events to create reliable models. Those events or event groups need to be well defined and timed with high precision in the same time-frame as the fMRI data.

In conclusion, PM demonstrated the ability to analyze and systematize behavioral patterns in complex semi-naturalistic behavior during gameplay and provided us with a wider overview on what was happening during gameplay by mapping the player's behavior. PM also gave us the possibility of inspecting different characteristics of the behavioral cycle. Finally, we can also recommend the use of PM for the description of such patterns and comparing them with patterns in populations with psychiatric disorders. This has potential to reveal behavioral differences in the future and to investigate therapeutic targets for psychiatric impairments.

## ACKNOWLEDGMENT

This work was supported by the German Research Foundation (DFG; IRTG 2150), the Federal Ministry of Education and Research (BMBF; 01EE1405A-C), and a fellowship of the German Academic Exchange Service (DAAD) to S.R.

## REFERENCES

[1] ESA, "Essential Facts About the Computer and Video Game Industry - 2017 Sales, Demographic and Usage Data," *ESA Rep. 2017*, 2017.

[2] K. Mathiak and R. Weber, "Toward brain correlates of natural behavior: fMRI during violent video games," *Hum. Brain Mapp.*, vol. 27, no. 12, pp. 948–956, 2006.

[3] M. Palaus, E. M. Marron, R. Viejo-Sobera, and D. Redolar-Ripoll, "Neural Basis of Video Gaming: A Systematic Review," *Front. Hum. Neurosci.*, 2017.

[4] P. Vorderer and J. Bryant, *Playing video games: Motives, responses, and consequences*. 2006.

[5] S. L. Smith, K. Lachlan, and R. Tamborini, "Popular Video Games: Quantifying the Presentation of Violence and Its Context," *J. Broadcast. Electron. Media*, 2003.

[6] K. M. Hullett, "The science of level design: Design patterns and analysis of player behavior in first-person shooter levels," 2012.

[7] R. Weber, K. M. Behr, R. Tamborini, U. Ritterfeld, and K. Mathiak, "What do we really know about first-person-shooter games? An event-related, high-resolution content analysis," *J. Comput. Commun.*, 2009.

[8] T. Dietz, "An examination of violence and gender role portrayals in video games: Implications for gender socialization and aggressive behavior," *Sex Roles*, 1998.

[9] K. M. Thompson and K. Haninger, "Violence in E-rated video games," *J. Am. Med. Assoc.*, 2001.

[10] W. M. P. Van der Aalst, B. F. Van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters, "Workflow mining: A survey of issues and approaches," *Data and Knowledge Engineering*. 2003.

[11] W. M. P. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. 2014.

[12] W. Van Der Aalst *et al.*, "Process mining manifesto," in *Lecture Notes in Business Information Processing*, 2012.

[13] A. Karla and A. De Medeiros, "ProM Framework Tutorial," *Eindhoven Univ. Technol. Eindhoven Netherlands Novemb. 2006*, 2008.

[14] M. R. Ma'Arif, "Revealing daily human activity pattern using process mining approach," in *International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, 2017.

[15] W. M. P. van der Aalst, B. F. van Dongen, C. W. Günther, A. Rozinat, E. Verbeek, and T. Weijters, "The Process Mining Toolkit," *Bus. Process Manag. Demonstr. Track (BPMDemos 2009)*, 2009.

[16] A. J. M. M. Weijters and J. T. S. Ribeiro, "Flexible heuristics miner (FHM)," in *IEEE SSCI 2011: Symposium Series on Computational Intelligence - CIDM 2011: 2011 IEEE Symposium on Computational Intelligence and Data Mining*, 2011.

[17] C. W. Günther and W. Van Der Aalst, "International conference on business process management," in *"Fuzzy mining–adaptive process simplification based on multi-perspective metrics.,"* 2007.

[18] C. J. Turner, A. Tiwari, and J. Mehnen, "A genetic programming approach to business process mining," in *Proceedings of the 10th annual conference on Genetic and evolutionary computation - GECCO '08*, 2008.

[19] L. Helplessness, "Learned helplessness martin," *Annu. Reviiew Med.*, 1972.

[20] C. S. Carver and T. L. White, "Behavioral Inhibition, Behavioral Activation, and Affective Responses to Impending Reward and Punishment: The BIS/BAS Scales," *J. Pers. Soc. Psychol.*, 1994.

[21] J. J. Allen, C. A. Anderson, and B. J. Bushman, "The General Aggression Model," *Current Opinion in Psychology*. 2018.

# Estimating the Impact of Incidents on Process Delay

Felix Mannhardt
Technology Management
SINTEF Digital
Trondheim, Norway
Email: felix.mannhardt@sintef.no

Petter Arnesen
Mobility and Economics
SINTEF Building and Infrastructure
Trondheim, Norway
Email: petter.arnesen@sintef.no

Andreas D. Landmark
Technology Management
SINTEF Digital
Trondheim, Norway
Email: andreas.dypvik.landmark@sintef.no

***Abstract*—Process mining reveals how processes in organisations are actually performed and pinpoints deviations from the desired process execution. Process delay is one type of deviation that can be detected. Specific activities may take longer than expected or the waiting times between activities may deviate from service agreements. However, the quantification of processing or waiting times is often only the starting point in identifying the underlying root causes for process delay. One such root cause are adverse incidents in the environment of the process such as malfunctioning of supporting systems or unavailability of resources. Data about these external factors is often neither included in the event log nor recorded precisely enough to be directly linkable to a specific set of process instances. This paper presents a method for estimating process delay caused by incidents for which only the approximate occurrence time is known. We link incidents that are recorded in an incident log to process delay and calculate the effect of incidents on process delay using a Markov chain Monte Carlo sampling (MCMC) approach. Our proposed method was evaluated in a project conducted with the infrastructure manager of the Norwegian railway system. We applied it to a large event log of more than 120 million events capturing block-level movements of trains in the railway network and estimated the impact on process delay of about 50 000 infrastructure-related incidents. This showed that the method is useful for providing decision support and insights on the effects of maintenance. Since then the method has become part of the standard toolbox of the infrastructure manager.**

## I. Introduction

While a process can be viewed as a sequence of events, with timings and ordering, that can be investigated with process mining methods [1]; in most processes it is also fruitful to discuss factors that negatively impact the process execution. These are often colloquially referred to as adverse events. These events are not necessarily a part of the process, but occurrences that directly or indirectly affect activities of a process. Statistical process control generally distinguishes between common-cause variations and special-cause variations [2], which correspond well to internal and external factors.

Common-cause variations, often internal to the process, are events caused by phenomena constantly active within a system, they can be probabilistically predicted, or simply seen as the natural variation or noise within a system. Typical common-cause variations are variable process performance due to capacity problems or difference process variants. Special-cause variations, often external to the system, are unpredictable and, even when internally sourced, variations outside the historical evidence base. Concrete examples are power outages, extreme weather conditions, etc. The latter source of delay often has an indirect causal connection with the execution of process activities. Therefore, the attribution of delay (or establishing a more direct cause-and-effect relation) in which the special-cause variation is isolated from the common-cause variation is often tricky.

We address the problem of estimating the impact of incidents on the performance of a process (special-cause variations) based on an event log, as well as a separate, non-integrated, incident log. The event log stores the process execution in terms of activity sequences and their timestamps as usually assumed in process mining. The incident log is a supplementary data source storing information on the occurrence of incidents with a possible influence on the process. Such incident logs, however, are likely not to contain precise information on the exact boundaries of an incident's influence on the process. Often, incidents are manually recorded after the fact or the exact time boundaries of their influence on the process are unknown.

Our contribution is an estimation method for process delay caused by external incidents. First, our method quantifies process delay based on comparing process performance to the typical performance as observed before and after the incident in an event log. Then, it connects this delay to an incident based on a Markov chain Monte Carlo (MCMC) estimation of the time window in which the incident influenced the process performance. The proposed method was evaluated in depth using a large event log obtained from railway traffic in the Norwegian railway network and corresponding incidents from a maintenance management system. In this scenario, estimation of process delay is necessary as the maintenance system contains only unreliable information on the exact occurrence time frame of incidents making it impossible to directly link the incidents to delay in the process.

This remainder of the paper is structured as follows. Section II expands on our motivation to study the delay estimation problem and illustrates it using a railway network process scenario; Section III presents the process delay estimation method; and Section IV describes the evaluation. Finally, Section VI concludes with an outlook on future work.

## II. Motivation

Figure 1 illustrates the process delay estimation challenge. Often the execution of a *process* is logged in an *event log*. This

978-1-7281-0919-0/19/$31.00 ©2019 IEEE
DOI 10.1109/ICPM.2019.00018

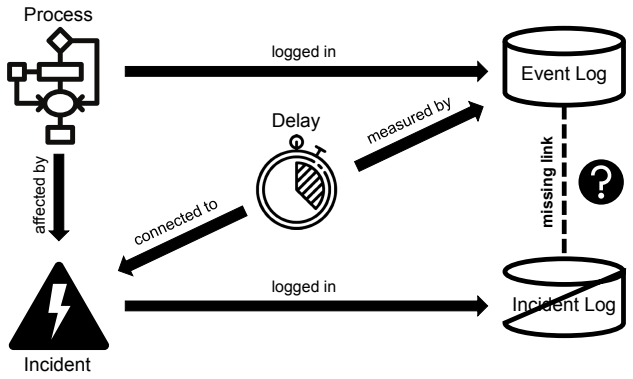



Fig. 1. Incidents affect the process execution and may cause process delay, which can be measured from the event log of a process. Incidents are also logged in an incident log, but cannot be directly linked to the events due to imprecise recording of the exact boundaries of an incident.

allows us to measure the performance of individual process instances or even specific process activities. By comparing to the normal process performance or to service agreements it is possible to measure process *delay*. Condition monitoring data provides information on adverse *incidents* that may affect the process performance. However, one challenge with such *incident logs* is often a weak or non-existent causal relation to operational performance metrics of the supported process. Establishing a causal relationship between the condition or state of physical infrastructure and the performance in use is impossible without perfect information. The goal of our work is to estimate this missing link between event log and incident log and connect individual incidents to the observed process delay.

We illustrate the problem of process delay estimation using an example taken from the railway domain in which adverse incidents on the railway infrastructure delay the block level movements of trains throughout the railway network. Railways are, from a statistical process control perspective, a process only barely in control. Schedules is an idealistic plan, and performance will usually be worse that planned. Hence, running with delays is the common case. This makes it extra difficult to determine and attribute *additional* lapses in performance (e.g. delays) over the common performance on the instance level. Common sources for process delays are passenger-related, operator-related, and infrastructure-related delays. These also interact through multiple complex pathways. So, the perfect separation of causality is assumed impossible. Moreover, the source of delay are interdependent, with several paths of correlation between them, necessitating an estimation-based approach to delay-allocation. Note that the estimation method is not limited to physical infrastructure as adverse incidents may also affect resources required in more traditional business processes such as loan applications or call centre handling.

Looking at the railway scenario from a process mining perspective, each run of a train through the network is a process instance and the block-level movement between stations can be

TABLE I
AN EXAMPLE RAILWAY TRAFFIC CONTROL EVENT LOG TAKEN FROM THE NORWEGIAN RAILWAY NETWORK WHICH IS USED AS INPUT TO OUR METHOD. TABLE ADAPTED FROM [3].

| id | train | time | schedule | station | track | type |
|---|---|---|---|---|---|---|
| $e_1$ | 407 | 06:30:43 | 06:30:00 | OPD | - | arrival |
| $e_2$ | 407 | 06:47:52 | 06:45:00 | OPD | OPD-FGH | departure |
| $e_3$ | 407 | 06:53:27 | 06:51:30 | FGH | OPD-FGH | arrival |
| $e_4$ | 407 | 06:53:49 | 06:52:00 | FGH | FGH-UBG | departure |
| $e_5$ | 407 | 07:02:49 | 07:00:30 | UBG | FGH-UBG | arrival |
| $e_6$ | 407 | 07:03:25 | 07:01:00 | UBG | UBG-BAK | departure |
| $e_7$ | 407 | 07:11:19 | 07:08:00 | BAK | UBG-BAK | arrival |
| $e_8$ | 407 | 07:13:15 | 07:09:00 | BAK | - | departure |
| $e_9$ | 42 | 09:30:03 | 09:29:00 | BAK | - | arrival |
| $e_{10}$ | 42 | 09:31:54 | 09:30:00 | BAK | BAK-UBG | departure |
| ... | ... | ... | ... | ... | ... | |

TABLE II
AN EXAMPLE INCIDENT LOG TAKEN FROM THE NORWEGIAN RAILWAY NETWORK WHICH IS USED AS INPUT TO OUR METHOD.

| id | object | location | time |
|---|---|---|---|
| $i_1$ | EH-MAS-123 | km 453 | 2017-12-04 06:58:00 |
| $i_2$ | SA-DRV-123 | km 442 | 2017-12-06 13:23:00 |
| $i_3$ | SA-SIK-123 | km 320 | 2017-12-12 20:51:00 |
| ... | ... | ... | ... |

modelled as process activities. In most railway networks, train movements are recorded in railway traffic control logs such as the one in Table I. Traffic control logs contain the running time of trains between stations and the dwelling time on stations. Based on this data, an event log suitable for measuring process delay can be built [3], [4]. We consider the movement between stations (column *track*) as process activity, which results in a process as depicted in Figure 2. Activities represent the movement of trains between stations (or intermediate measurement point) and the time spent between activities (shown on the edges) is the dwelling time on a station. Then, we can compute the delay by comparing the actual running time with the scheduled running time or the average running time in the past days.

Next to traffic control logs, the railway Infrastructure Manager (IM) manages all infrastructure condition related events and work orders in a maintenance management system. From this system an incident log as shown in Table II can be extracted. Each incident relates to a certain *object* that can be associated to a specific track by its location. The time at which the incident occurred is recorded, but due to manual registration often it is only an approximation of the actual time. Specifically, the time boundaries in which the incident may have had a influence, i.e., from its occurrence until the incident was resolved, are not recorded or not available with sufficient quality. Given this data the IM wants to connect process delay (i.e. late trains) to specific incidents in order to make informed decision on maintenance investments.

## III. DELAY ESTIMATION METHOD

The proposed process delay estimation method infers the missing link between occurrences of *adverse incidents*, for which the exact time boundaries are unknown, and *process*

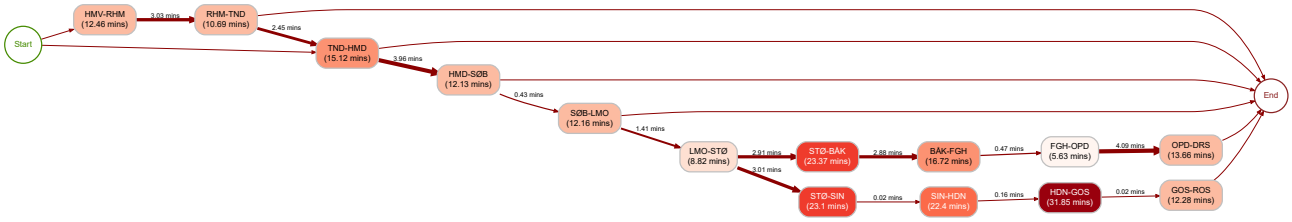



Fig. 2. Process map (directly-follows relations) of a small part of the Norwegian railway network discovered using half a year of traffic control logs for trains riding from Trondheim towards Oslo when using 90% of the most common traces. Note that on some stations trains do not stop, which explains the very low time waiting at some station (edges).

*delays* as measured from an event log. First, we describe the inputs and assumptions of our method. There are three inputs:

- an event log $L$ of the process,
- a log $I$ of adverse incidents;
- a relation $I2A$ connecting incidents to activities.

An event log $L$ is defined over a set of unique events $E$ and a set of activities $A$. It consists of traces $s \in L$ that are sequences of events, i.e., $L \subseteq \mathbb{P}(E^*)$ such that each event $e \in E$ occurs in one and only one trace. Each event $e \in s$ corresponds to the occurrence of some distinct activity from a pre-defined set of activities $act_E(e) \in A$ together with two timestamps: the start time of the activity $time^s_E(e) \in \mathbb{N}$ and the completion time of the activity $time^c_E(e) \in \mathbb{N}$. Such event log is also denoted as *activity log* as the two standard life-cycle transitions *start* and *complete* are combined in one event. Generally, non-compliant event logs can be pre-processed in an activity-log format even in the presence of noise, e.g., by using alignment-based conformance checking [6]. For convenience, we define $dur_E(e) \in \mathbb{N}$ with $dur_E(e) = time^c_E(e) - time^s_E(e)$ to return the total duration of an activity execution.

The set of potential adverse incidents $I$ contains all those incidents which may have an impact on the process execution. For each incident $i \in I$ its time of occurrence: $time_I(i) \in \mathbb{N}$ is recorded. Note that this recording may have been done manually and, thus, could be not reliable, e.g., the incident may actually have occurred before this time.

The incident-activity relation $I2A \subseteq I \times A$ connects each incident to the activity most likely affected by the incident. We required this relation to avoid spurious delay registration and focus on activities for which we can assume a likely causal relation between the incident and the process performance. Obtaining $I2A$ is not always straightforward, but often it can be established automatically, e.g., in our railway scenario we can use available documentation to relate the location of objects for which an incident is registered to the track on which the train rides. Similarly, existing process models used to determine for which activities resource affected by an incident are required.

Taking this input, our method computes the delay attributable to each incident, i.e., $D \subseteq I \times \mathbb{Q}$. We determine the delay using the following four steps for each incident $i \in I$ and its set of related process activities $A_i = \{a \mid (i,a) \in I2A(i)\}$:

1) we estimate the normal performance of the process activity;
2) we classify process activity instances around the approximate time of incident into three classes: severely delayed, delayed, normal;
3) we use MCMC sampling to determine in which time frame and how likely the process performance was influenced by the incident;
4) we accumulate the difference between the normal process performance and the observed performance for all process instances executing the activity within the estimated time frame.

In the next four sections, we describe each step in detail for a single incident and its related process activity.

### A. Estimating the normal process performance

First, we need to determine the normal performance of the process without influence of an incident. As our definition of process performance is relative to the activity which is considered to be connected to the incident through some causal link (required resource etc.), we primarily consider the local process performance in terms of the service time of that activity.

There are several ways to estimate what is considered normal. We determine the normal process performance based on the average performance in the time preceding and succeeding the incident. Our proposal does not make assumptions on Service Level Agreements (SLA) as we aimed to also identify the impact of incidents on process delay even when the delay is within a given SLA. This is important when the delay estimation is to be used for a process-improvement scenario since problems need to be identified before a SLA is violated.

Assume that incident $i$ occurred at time $\mu_{t,i} = time_I(i)$. To determine the normal performance, we take the *average* and *standard deviation* of the process performance for each activity $a \in A_i$ observed in the time windows $[\mu_{t,i} - t_{N2}, \mu_{t,i} - t_{N1}]$ and $[\mu_{t,i} + t_{N1}, \mu_{t,i} + t_{N2}]$. Moreover, we exclude outliers with a very high or low duration. For example, in our use case, we excluded some unusually slow freight trains, which appeared regardless of incidents.

Given an event log $E$, its set of traces $L$, and the process activity $a \in A_i$ of interest. Let $E^a_{std} \subset E$ be the set of events

observed in the *normal state* time window:

$$E^a_{std} = \{e \in E \mid act_E(e) = a \wedge (time_E(e) \in [\mu_{t,i} - t_{N2}, \mu_{t,i} - t_{N1}] \vee time_E(e) \in [\mu_{t,i} + t_{N1}, \mu_{t,i} + t_{N2}])\}$$

We compute the average $\mu^a_{std}$ and standard deviation $\sigma^a_{std}$ as usual:

$$\mu^a_{std} = \frac{\sum_{e \in E^a_{std}} (dur_E(e))}{|E^a_{std}|} \quad (1)$$

$$\sigma^a_{std} = \sqrt{\frac{\sum_{e \in E^a_{std}} (dur_E(e) - \mu^a_{std})^2}{|E^a_{std}| - 1}} \quad (2)$$

There are several considerations to be taken into account when choosing $t_{N1}$ and $t_{N2}$. Parameter $t_{N1}$, the time distance from the assumed time of the incident, needs to be large enough so that the incident cannot possibly have had an influence. Similarly, $t_{N2}$ should be large enough so that the effect of unrelated delays before or after the incident on the normal process performance estimate is negligible. For example, in our case we choose $t_{N1}$ to be 24 hours and $t_{N2}$ to be three days.

### B. Classifying the duration of activity executions

Having determined the normal process performance in a standard situation, for each activity $a \in A_i$ we build a sorted sequence of events $seq^i_E(a) : A \to E^*$ from the executions of activity $a$, which are recorded in the event log in the time window $[\mu_{t,i} - t_{N1}, \mu_{t,i} + t_{N1}]$ directly around the suspected time of the incident:

$$seq^i_E(a) = \langle e_1, \ldots, e_n \rangle \; s.t. \; \forall_{1 \le j < k \le n} ( \{e_j, e_k\} \subset E \wedge a = act_E(e_j) = act_E(e_k) \wedge t_j = time^c_E(e_j) \wedge t_k = time^c_E(e_k) \wedge t_j < t_k \wedge \{t_j, t_k\} \subset [\mu_{t,i} - t_{N1}, \mu_{t,i} + t_{N1}])$$

Hereafter, we simply refer to this sequence as $s_{a,i}$, i.e., $s_{a,i} = seq^i_E(a)$.

We classify each event $e \in s_{a,i}$ into one of three classes: $C = \{0, 1, 2\}$ using function $\gamma : E \times A \to C$ with:

$$\gamma(e, a) = \begin{cases} 1, & \text{if } dur_E(e) < 1.5\,\sigma^a_{normal} + \mu^a_{normal} \\ 2, & \text{otherwise} \\ 3, & \text{if } dur_E(e) > 3\,\sigma^a_{normal} + \mu^a_{normal} \end{cases}$$

Class 1 are **normal** instances in which the execution time is in the standard range for activity $a$ that was observed before and after the incident. Class 2 are **delayed** instances which considerably longer. Class 3 are **severely delayed** activity executions with an unusually long execution time.

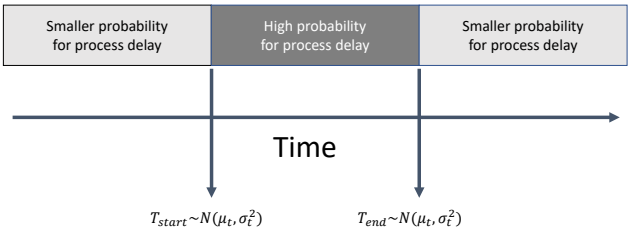



Fig. 3. We assume that the probability for process delay is increased in a time frame around the approximate time of the incident.

### C. Estimating the influence of an incident

Now we have all the ingredients to estimate the influence on the execution time of process activities. As illustrated in Figure 3, we denote with $t_{start}$ and $t_{end}$ the time window in which incident $i$ had an influence on activities $A_i$, which we want to estimate. We will use the notation $P(\cdot)$ for probability distributions, and assume apriori for each $a \in A_i$ that the probability distribution to register an event $e_j \in s_{a,i}$ from event log $E$ at time $t_j = time^c_E(t)$ as being classified with class $\gamma(a, e_j) = c_j$ outside of the influence of the incident $t_j < T_{start}$ or $t_j > T_{end}$ has a discrete probability distribution $P_0(c_j|p_0)$. Conversely, the probability distribution for an activity execution in which $T_{start} < t_j < T_{end}$ is $P_1(c_j|p_1)$.

Parameters $p_0$ and $p_1$ are case dependant parameters that indicate the probabilities to observe the three different classes of delay. For example, in our specific case we choose the parameters as $p_0 = (0.94, 0.055, 0.005)$ and $p_1 = (0.93, 0.06, 0.01)$. In word, we assume apriori that there is a slightly higher probability to observe delayed or severely delayed events within the time boundaries in which the incident was influential on the system. These parameters were in our case tuned by running several pilot estimations and compare the results to cases annotated by experts from the Norwegian railway IM.

As an alternative to manual parameter tuning, these parameters could also be estimated by assigning a less informative prior distribution to the parameters $p_j$ for $j = 0, 1$. For instance using a Dirichlet distribution as in [5]. However, simulating them along-side the other parameters in the posterior, would be much more computationally expensive, requiring all incidents to be considered in one long MCMC run. Therefore, in this first approach to solving this problem we propose tuning these parameter values base on problem experience and a smaller more know data. Or as another alternative $p_j$ for $j = 0, 1$ could be estimated from a separate dataset including both incident-free or known incident cases. For other problems, directly assigning vaguer priors might also be a possibility that works well, however this approach was not investigated in this paper but will be subject to further research.

For each sequence $s_a$ of events we assign a parameter $IO_a = 0, 1$ to denote whether or not the sequence of events related to activity $a$ is not affected by the incident or affected by the incident, respectively. To this parameter we assign

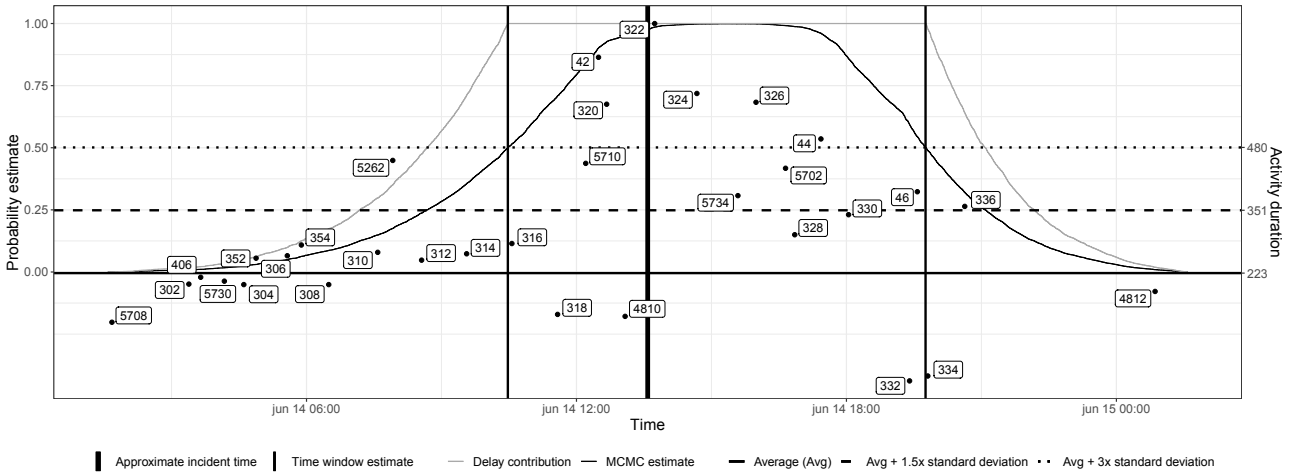



Fig. 4. Example of the estimation of the influence of an incident on activity durations of one specific activity. Each dot represents an activity execution, which is a train running along a certain track. The thick black vertical line depicts the assumed time at which the incident occurred. The horizontal lines show the estimates of the normal performance $\mu^a_{std}, \mu^a_{std} + 1.5\sigma^a_{std}$, and $\mu^a_{std} + 3\sigma^a_{std}$. Our simulation results in the black curve that shows the probability according to which a certain timestamp was placed between $T_{start}$ and $T_{end}$. The grey line depicts the factor with which a prolonged activity duration contributes to the calculated process delay.

apriori the distribution $P(IO_a) = 0.5$ for $IO_a = 0, 1$. For the start and end times we assume apriori that $T_{start}, T_{end} \sim N(\mu_{t,i}, \sigma_t^2)$ with $T_{start} < T_{end}$, where $\mu_{t,i}$ is the assumed time of the incident and $\sigma_t = 4$ is a case dependant parameter. By Bayes theorem the posterior distribution for $T_{start}$ and $T_{end}$ is

$$P(T_{start}, T_{end}|E, A_i, IO, p_0, p_1, \mu_{t,i}, \sigma_t) \propto$$
$$I(T_{start} < T_{end}) P(T_{start}|\mu_{t,i}, \sigma_t) P(T_{end}|\mu_{t,i}, \sigma_t) \cdot$$
$$\prod_{a \in A_i} P(IO_a) \left( IO_j \prod_{e_j \in s_a} [g_1(j) + g_0(j)] \right),$$

where $I(\cdot)$ is the identity function being 1 if the argument is true and 0 otherwise, and

$$g_1(j) = I(t_j \in [T_{start}, T_{end}]) P_1(c_j|p_1),$$
$$g_0(j) = (1 - I(t_j \notin [T_{start}, T_{end}])) P_0(c_j|p_0).$$

In our use case we simulate 20000 iterations using the Metropolis-Hastings algorithms using multiple different proposal distributions for $T_{start}$ and $T_{end}$ and estimate the marginal aposterior distributions $P(T_{start}|E, A_i, IO, p_0, p_1, \mu_{t,i}, \sigma_t)$ and $P(T_{end}|E, A_i, IO, p_0, p_1, \mu_{t,i}, \sigma_t)$. This gives us an estimation of the most probable start and end time of the influence that the incident had on process delay.

### *D. Attributing process delay to an incident*

Figure 4 gives an example of how the proposed estimation method works for a single activity when being applied to an event log in which activities instances are the trains driving between stations. It shows an excerpt of the registered activity durations obtained from the Norwegian railway IM for a single activity (i.e., single track) from several process instances (individual trains). Even though the registered time for the incident was around 13:30, there was already some significant delay before that time. The black curve indicates the probability according to which a certain timestamp was placed between $T_{start}$ and $T_{end}$.

The delay for a single event $e$ can be computed by comparing the recorded activity duration to the average duration in a normal situation, i.e., $delay_a(e) = dur_E(e) - \mu^a_{std}$. However, there are several ways to accumulate the individual delay to the total process delay attributable to the incident. A straightforward method is to multiply the delay recorded by event $e$ with the probability estimate at time $time^c_E(e)$. We found in the validation of our case that this often lead to an under-approximation of delay. Also, in practice a delay is either caused by an incident or not caused by an incident. Therefore, directly using the simulation results can be problematic for interpretability of the results by process stakeholders. Thus, we adopted the following method.

We fully count the delay for every event for which the probability estimate is at least 50%, i.e., in at least half of the simulated samples $time^c_E(e)$ falls in the window $[T_{start}, T_{end}]$. This captures the intuition that in a core time frame the delay of each activity instance is fully attributed to the incident. For the remainder of the activity instances, we add only a fraction of the delay since the influence of the incident is less obvious. This yields the grey curve in Figure 4 that we obtain by scaling the simulation results with a factor of 2 and counting all delay caused by events that occur at a timestamp for which this scaled probability estimate exceeds 1.0 in full. For example, the delay of the process instance 46 is counted fully, whereas the delay for the process instance 336 is discounted with a factor of approximately 0.25.

In summary, we compute the total delay $d_E(i) \in \mathbb{Q}$ for

incident $i$ connected to activities $A_i$:

$$d_E(i) = \sum_{a \in A_i} \sum_{e \in seq_E^i(a)} (delay_a(e) \cdot min(1, 2 \cdot p_e))$$

where

$$p_e = P(time_E^c(e) \in [T_{start}, T_{end}] | E, A_i, IO, p_0, p_1, \mu_{t,i}, \sigma_t)$$

based on the simulation.

We repeat the procedure and obtain the accumulated delay $d_i$ for each incident $i$ and build the set $D = \{(i, d) \mid i \in I \wedge d = d_E(i)\}$.

## IV. Case: Norwegian railway traffic

We validated the proposed method using the process of train traffic in the Norwegian railway network. Our data source is a large event log from the traffic control system as well as the corresponding infrastructure incident log.

Norwegian railway infrastructure is owned and managed by a single Infrastructure Manager (IM). The IM is in charge of the infrastructure, including monitoring infrastructure condition, incident handling and preventative and corrective maintenance. The operation of rolling stock on the infrastructure is an open market, and currently there are 10 operators with a license for traffic on the general infrastructure. The majority of the infrastructure is a single-track infrastructure (94% single track, 65% electrified), in a star-shaped network around the capital. The topology and restrictions in single-track operation means that adverse incidents and their corrective maintenance is a important limiting factor for railway performance.

Infrastructure condition related events and work orders reside in a traditional maintenance management system, whilst the traffic data (i.e. system performance metrics) resides in an event log which automatically samples the track-side train detection in order to provide automatic second-resolution punctuality information. Prior to the algorithm there were no automatic allocation of delays, but only manual attribution. This attribution was conducted by dispatchers, acting on information conveyed by multiple sources (telephone, status logs, etc.). In a pre-study we identified that this manual attribution has obvious signs of batching, missing or obviously erroneous attribution.

### A. Set-up

We implemented the proposed method in R and applied it to a large event log with more than 110 million events describing block-level movements of trains that was obtained from the traffic control system of the IM. Whilst being a centrally controlled system, the event log data was of varying quality which led to challenges similar to more traditional business process mining scenarios. For example, some of the tracks are still manually recorded. Such manual recording, data transfer issue, and non-standard train traffic caused only partially ordered events, swapped events, and several other issues. We used knowledge on the actual infrastructure, i.e., the de-jure process model, to mitigate such problems similar to conformance checking methods [6].

We obtained the incident-activity relation for 50 000 geo-located incidents that were recorded in the maintenance management system by using the location of the incident and lookup of the two closest stations to each side of the track. We excluded incidents that evidently cannot affect the performance of the process, e.g., incidents regarding the displays on a station.

For the tuning of parameters ($p_0$ and $p_1$), we observed that they are sensitive to large changes, however, small changes in parameters do not change the result significantly. Thus, the most important lesson when choosing these parameters is that an occurrence of a delay is only slightly more probable under an active incident, the relative difference here being more important than the actual level on the probabilities.

We repeated the estimation 20 times for each incident using the parameters indicated when presenting the method and compared the results to evaluate whether the MCMC simulation converged to a stable solution. Also, we investigating a large number of trace-plots for all simulated parameters, that showed convergence after the proposed number of iterations.

With repetitions, the computation took on average 6 minutes for a single incident including data retrieval on a single computation thread using standard server hardware. Note that the calculation can be trivially parallelised which brings the computation time down a level that is feasible in practice. The results were written back to the data warehouse environment of the IM for reporting and planning purposes.

### B. Validation

In lack of a ground truth, we validated the results by comparing the results to a database of delays that were manually allocated to some incidents by train operators. Among the process instances for which manual registration data was available (only about 10% of the total data), we found that in most cases (74%) our approach found some or even all the delays manually allocated. Surprisingly, we found that in 36% of the cases, we did not find any of the delay attributed to incidents manually. We conducted an interview with a train operator to clarify the difference. Several possible reasons for this discrepancy were identified, which indicate the differences are mainly due to data quality issues.

The primary reason is that, due to manual nature of the delay registration and batch registration after a busy shift, sometimes trains are registered that have been already delayed before reaching the position of the incident (i.e., before executing the process activity in question). Moreover, we found that in some parts of the railway network the data quality of the traffic registration is poor, which results in missing delay as our proposal only takes the existing data into account. Another reason are knock-on effects of delayed trains to other trains in the railway network. These effects are not captured by this proposal. In practice, we already integrated the knock-on effect estimation approach presented in [7] for the single-track part of the network. However, we consider this as out of scope for this work.

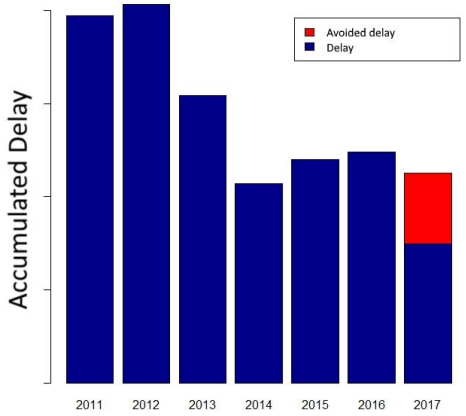



Fig. 5. Accumulated delay for a specific part of the Norwegian railway network as estimated based on the proposed method. The red bar in 2017 indicates the likely impact of incidents that were avoided by a preventive maintenance system. The y-axis has been concealed due to confidentiality.


Fig. 6. The analysis dashboard built on top of the delay estimation data.

### C. Results

There are many opportunities to use the process delay data produced by our method. We present two use cases for the data produced on the Norwegian railway traffic process.

In the first use case, we leverage the fact that it is possible to automatically obtain a large amount of delay data and that this data can be connected to a single resource in the infrastructure. In this way, it is possible to get a reliable estimate of the typical delay effect on the process that originates from a specific failing resource. Recently, the Norwegian IM has invested in preventive maintenance solutions for several heavily used turnouts. Upgraded turnouts raise an alarm when the equipment is likely to fail. We used data on these alarms, which can be seen as prevented incidents, and the aggregated median estimated process delay from the years beforehand to quantify the delay-reduction effect of this investment in predictive maintenance. The bar plot in Figure 5 shows the estimated savings in term of prevented delay. Such insights can be very valuable for evidence-based decision support.

In the the second use case, we used the unprecedented scale of the delay database, in comparison to manual registration, to open up new possibilities for exploratory data analysis and reporting. For this purpose, we built an analysis dashboard that can be used by the Norwegian IM. The dashboard has been used in several workshops and is in ongoing use to identify reoccurring issues with infrastructure and their effect on delays. A distinct advantage of this fully data-driven approach is that also the effect of many small impacts on process delay can be quantified. Whereas the reasons for failures resulting in rare large delays are well-known and can be manually investigated, the effect of many small incidents is difficult to establish but may well contribute more towards the overall customer satisfaction.

## V. Related work

Investigating the performance of processes using event logs is one of the main tasks of process mining. A seminal work in this regard is [8] which establishes the notion of alignments to project reliable performance information extracted from event logs on process models. In [9] the observed performance of a process is captured with stochastic Petri nets. Decision trees are used in [10] to analyse influential factors of business process performance. Differently to our method, these factors are assumed to be internal to the process and, thus, already integrated with the process events. In [11] the correlation between resource workload and processing speed in a process is investigated based on an event log, however, no process delay is quantified.

Another type of methods aims to predict the process performance (e.g., remaining time, time to next activity). Like our setting the work in [12] considers not only intra-case features but takes a global view on all running cases. However, external incidents are not considered. In [13] the process prediction considers the influence of an environmental state, which is similar to our use of the incident log as external factor but with the aim to predict performance.

Besides these process-model-based approaches, there are also several proposals to investigate the process performance using visual analytics. A basic technique for comparing the performance across several traces is the Dotted Chart [14] in which events are visualised on a 2D scatter plot. Two more recent proposal are the ProcessProfiler3D [15], in which a general framework for 3D visualization of process performance is proposed, and the Performance Spectrum Miner [16], which focuses on visualizing performance variability over time. While useful for exploratory analysis of causes for process delay, it is not automatically attributing delay to incidents.

Regarding our case study, in railway research the effect of unplanned events such as infrastructure and rolling stock malfunctions, passenger-related incidents or and inclement weather is well known [17]. Interdependencies between in-

dividual delays and, so called, knock-on delays have been studied [7], [18]. Also, there is some research that took a process mining perspective on railway operations [3], [4], [19]. However, to the best of our knowledge, there is no work trying to quantify the effect of incidents on the overall delay of a process based on events logs.

## VI. Conclusion

The process-oriented view on event data in process mining has led to a large number of methods that reveal insights about process execution. However, there is relatively little work on the impact that external factors have on processes even though processes are seldom executed in isolation, and events that are not directly related to the execution itself may have a large impact on it. Such *adverse incidents* are often also registered, but rarely in the same system, sometimes only manually, and often with considerable uncertainty regarding the exact time frame in which the incident had influence on the process. Thus, it is difficult to quantify their impact on the process execution.

The main contribution of our work is a method for process delay estimation caused by adverse incidents, which uses process event logs and some minimal approximate information on the incidents. We use Markov chain Monte Carlo (MCMC) sampling to determine the most likely times at which the influence of an incident on process performance started and ended. Then, we use the estimated likelihoods to accumulate the overall delay, i.e., difference in performance from a normal situation, that can be attributed to the incident. We implemented and evaluated the method using a large database of incidents and railway traffic control logs in the Norwegian railway system between 2011 and 2018. The evaluation showed that the method can reliably assign process delay to incidents and thta it is useful for showing the effect of investments in predictive maintenance. Finally, our method puts focus on the overall effect of many minor infrastructure failures individually resulting in small delays, which otherwise would be difficult to establish.

There are several options to improve the proposed method by addressing its limitations in future work.

First, we aim to apply the method to non-infrastructure related processes. In the railway case the incident-activity relation was easy to obtain, but in a general business context it may be more difficult to determine this relation.

Second, multiple co-occurring incidents that affect the same activity pose an issue for the proposed method both due to possibly multi-modal delay distributions as well as the difficulty to divide the delay properly. This, as well as, knock-on and complex queuing effects are limitations of our method and areas for future work. Regarding knock-on effects we did initial research based on the method presented in [7]; however, this method is not straightforward to apply it to the case of generic business processes without additional assumptions.

Last, the Posterior distribution which is simulated using MCMC uses a defined set of parameter values in the prior distribution. These parameter values do not generalize to other problems, and would need to be calibrated in each case. An alternative that could be explored is to define a less informative prior distribution, for instance using hyper distributions and simulating parameters $p_0$ and $p_1$ as well, however this comes with a high computational price.

## Acknowledgment

This research was in part supported by Bane NOR.

## References

[1] W. M. P. van der Aalst, *Process Mining - Data Science in Action, Second Edition*. Springer, 2016.
[2] W. E. Deming, *Out of the Crisis*. Cambridge University Press, 1986.
[3] F. Mannhardt and A. D. Landmark, "Mining railway traffic control logs," in *21st EURO Working Group on Transportation Meeting, EWGT 2018*, ser. Transportation Research Procedia, vol. 37, 2019, pp. 227–234.
[4] P. Kecman and R. M. P. Goverde, "Process mining of train describer event data and automatic conflict identification," in *Computers in Railways XIII*. WIT Press, Sep. 2012.
[5] P. Arnesen, T. Holsclaw, and P. Smyth, "Bayesian detection of change-points in finite-state markov chains for multiple sequences," *Technometrics*, vol. 58, no. 2, pp. 205–213, 2016.
[6] J. Carmona, B. van Dongen, A. Solti, and M. Weidlich, *Conformance Checking*. Springer International Publishing, 2018.
[7] A. Ø. Sørensen, A. D. Landmark, N. O. Olsson, and A. A. Seim, "Method of analysis for delay propagation in a single-track network," *Journal of Rail Transport Planning & Management*, vol. 7, no. 1-2, pp. 77–97, Jun. 2017.
[8] W. M. P. van der Aalst, A. Adriansyah, and B. F. van Dongen, "Replaying history on process models for conformance checking and performance analysis," *WIREs Data Min Knowl Discovery*, vol. 2, no. 2, pp. 182–192, 2012.
[9] A. Rogge-Solti, W. M. P. van der Aalst, and M. Weske, "Discovering stochastic petri nets with arbitrary delay distributions from event logs," in *BPM 2013 Workshops*, ser. Lecture Notes in Business Information Processing, vol. 171. Springer, 2014, pp. 15–27.
[10] B. Wetzstein, P. Leitner, F. Rosenberg, I. Brandic, S. Dustdar, and F. Leymann, "Monitoring and analyzing influential factors of business process performance," in *2009 IEEE International Enterprise Distributed Object Computing Conference*. IEEE, Sep. 2009.
[11] J. Nakatumba and W. M. P. van der Aalst, "Analyzing resource behavior using process mining," in *Business Process Management Workshops*. Springer Berlin Heidelberg, 2010, pp. 69–80.
[12] A. Senderovich, C. D. Francescomarino, C. Ghidini, K. Jorbina, and F. M. Maggi, "Intra and inter-case features in predictive process monitoring: A tale of two dimensions," in *BPM*, ser. Lecture Notes in Computer Science, vol. 10445. Springer, 2017, pp. 306–323.
[13] F. Folino, M. Guarascio, and L. Pontieri, "Discovering context-aware models for predicting business process performances," in *OTM Conferences (1)*, ser. Lecture Notes in Computer Science, vol. 7565. Springer, 2012, pp. 287–304.
[14] M. Song and W. M. van der Aalst, "Supporting process mining by showing events at a glance," in *Workshop on Information Technologies and Systems (WITS'07)*, 2007, pp. 139–145.
[15] M. Wynn, E. Poppe, J. Xu, A. ter Hofstede, R. Brown, A. Pini, and W. van der Aalst, "ProcessProfiler3d: A visualisation framework for log-based process performance comparison," *Decision Support Systems*, vol. 100, pp. 93–108, Aug. 2017.
[16] V. Denisov, D. Fahland, and W. M. P. van der Aalst, "Unbiased, fine-grained description of processes performance from event data," in *BPM 2018*, ser. Lecture Notes in Computer Science, vol. 11080. Springer, 2018, pp. 139–157.
[17] N. O. Olsson and H. Haugland, "Influencing factors on train punctuality — results from some norwegian studies," *Transport policy*, vol. 11, no. 4, pp. 387–397, 2004.
[18] H. Flier, R. Gelashvili, T. Graffagnino, and M. Nunkesser, "Mining railway delay dependencies in large-scale real-world delay data," in *Robust and Online Large-Scale Optimization*, ser. Lecture Notes in Computer Science. Springer, 2009, vol. 5868, pp. 354–368.
[19] G. Janssenswillen, B. Depaire, and S. Verboven, "Detecting train reroutings with process mining," *EURO Journal on Transportation and Logistics*, vol. 7, pp. 1–24, Apr. 2017.

# Assessing Software Development Teams' Efficiency using Process Mining

João Caldeira
Fernando Brito e Abreu
José Reis
ISTAR-IUL
Instituto Universitário de Lisboa (ISCTE-IUL)
Lisboa, Portugal
{jcppc, fba, jvprs}@iscte-iul.pt

Jorge Cardoso
CISUC, Dept. of Informatics Engineering
University of Coimbra, Portugal
Huawei Munich Research Center, Germany
jcardoso@dei.uc.pt

***Abstract*—*Context.*** **Improving the efficiency and effectiveness of software development projects implies understanding their actual process. Given the same requirements specification, different software development teams may follow different strategies and that may lead to inappropriate use of tools or non-optimized allocation of effort on spurious activities, non-aligned with the desired goals. However, due to its intangibility, the actual process followed by each developer or team is often a black box.**

***Objective.*** **The overall goal of this study is to improve the knowledge on how to measure efficiency in development teams where a great deal of variability may exist due to the human-factor. The main focus is on the discovery of the underlying processes and compare them in terms of efficiency and effectiveness. By doing so, we expect to reveal potentially hidden costs and risks, so that corrective actions may take place on a timely manner during the software project life cycle.**

***Method.*** **Several independent teams of Java programmers, using the Eclipse IDE, were assigned the same software quality task, related to code smells detection for identifying refactoring opportunities and the quality of the outcomes were assessed by independent experts. The events corresponding to the activity of each team upon the IDE, while performing the given task, were captured. Then, we used process mining techniques to discover development process models, evaluate their quality and compare variants against a reference model used as "best practice".**

***Results.*** **Teams whose process model was less complex, had the best outcomes and vice-versa. Comparing less complex process variants with the "best practice" process, showed that they were also the ones with less differences in the control-flow perspective, based on activities frequencies. We have also determined which teams were most efficient through process analysis.**

***Conclusions.*** **We confirmed that, even for a well-defined software development task, there may be a great deal of process variability due to the human factor. We were able to identify when developers were more or less focused in the essential tasks they were required to perform. Less focused teams had the more complex process models, due to the spurious / non-essential actions that were carried out. In other words, they were less efficient. Experts' opinion confirmed that those teams also were less effective in their expected delivery. We therefore concluded that a self-awareness of the performed process rendered by our approach, may be used to identify corrective actions that will improve process efficiency (less wasted effort) and may yield to better deliverables, i.e. improved process effectiveness.**

## I. Introduction

Inaccurate planning and/or project plan deviations cause substantial financial losses on software development projects [1]. Further, constant inaccuracies and losses may degrade the reputation of development teams as they become perceived as non-compliant to organizational plans and budget forecasts.

Critical success factors have always been at the forefront of the research related with software development projects [2]–[5]. The existence of a vast literature about this topic, either about successes [1] or failures [6], [7], reveals the concerns and doubts that still haunt software development practitioners regarding the efficiency and effectiveness of their own projects.

It is frequently suggested that software projects can be assessed across four perspectives: quality, scope, time and cost [2], which are related with the planning and execution of the project's main activities. Each perspective has its own critical success and failure factors, that can be grouped into five different dimensions: organizational, people, process, technical, and project [3]. In this paper, we will be mainly concerned with the effect of the human factor in process variability.

To start a software development project from scratch is a complex activity on its own [8], specially in the absence of a formalized process or methodology [9] that acts as a referential. Evidences found suggest that in addition to the initial project planning, the way people are organized, the tools they use and the processes they follow are key features for the success or failure of any software project [9]. As for software development, although prescribed process models may exist, projects often do not comply with them, both because each developer or team usually has some freedom to interpret the process and because its compliance is not verified on the run, since it is mainly intangible. As a result, it has been noted that process executions (i.e. projects) often deviate from what was planned [10]. In this paper we bring further evidence that the human factor is a very important source of process variability and the latter will have an impact on process efficiency and effectiveness.

To understand how the process was actually performed by its practitioners, we used process mining techniques. Our

978-1-7281-0919-0/19/$31.00 ©2019 IEEE
DOI 10.1109/ICPM.2019.00020

approach, initially proposed in [11], captures events due to practitioners activities executed in the IDE, as well as records which artifacts were used and when, plus additional details on the ecosystem of components supporting the process. This new perspective on software development analytics, that uses process mining, allows the discovery of the actual processes practitioners are following, as well as deviations from those they were supposed to comply to, without the complexity and workload of collecting and merging information from different information systems, such as, source code systems, configuration management repositories or bug tracking tools. As we will show later in this article, we were able to identify the most and less efficient teams, and the ones that drifted less from the same process when executed by an expert.

This paper is organized as follows: on section II we introduce software development analytics challenges and introduce process mining as a natural option to mine software process events' logs; on section III we present the research questions, describe the experiment setup and the methods used for data analysis; next, on section IV, we present the results, elaborate on the main findings and identify threats to validity; finally, in section V, we draw the main conclusions and outline the future work.

## II. Context

### A. Software Development and the IDE

Nowadays, most software practitioners develop their work upon an IDE (Integrated Development Environment), such as `Eclipse`, `IntelliJ IDEA`, `Netbeans` or `Visual Studio Code`. To a greater or lesser extent, those IDEs support different software development life cycle activities, such as requirements elicitation, producing analysis and design models, programming, testing, configuration management, dependencies management or continuous integration. In this paper we will consider `Eclipse`, which owes its wide adoption to the vast plethora of plugins available in its marketplace. `Eclipse` distributions are customized for specific users / purposes, such as for modellers, programmers, testers, integrators or language engineers. Herein, we will consider the standard distribution, which is particularly suited to programmers.

An IDE, in addition to the artifacts it handles, contains metadata about the developers' activities that may reveal the reasons why some individuals and teams are more efficient than others. Moreover, it may have hidden in its usage, parts of the logic why some projects are successful and others fail. Those development activities can be identified by mining the large amount of events created during the execution of the IDE core components and the installed plugins.

### B. Process Mining Within the IDE

Process Mining is now a mature discipline with validated techniques producing accurate outcomes on several business domains [12], [13]. A process mining project, if best practices are followed [14], should use goals and event logs as inputs, and produces actions to implement as outputs. The goals may consist of improving some performance indicators, such as time, risks and costs associated to a specific process, or simply to maximize a service level. Actions may be the redesign of a specific project, adjust a current process or, if there is a fluctuation in case volume, one may want to include more resources.

Our short-term goal, whose fulfillment we will describe in this paper, was to assess teams' efficiency by mining the software development process flow and variability that occurs due to the human factor. Our medium-term goal is to provide operational support to software developers, systematically and continuously using current event data to recommend the best activity, adequate resource or action to execute now or in the future. In both cases we will take as input the events emerging from using the IDE. Those events convey a spaghetti-like process [15] mainly because there is a very large number of possible commands/tasks to execute within any IDE that will grow exponentially with the number of installed plugins and, as a consequence, so grows the potential complexity of any mined process.

### C. Related Work

This work is in the crossroads of software development practices and process mining techniques. Much have been said in literature about software development processes [16], [17] and process mining separately [18]. However, elaborating about works combining these two disciplines requires a careful approach, mainly because their intersection is vague in some cases and not fully explained in others. Going back almost a decade, [12] have mined software repositories to extract knowledge about the underlying software processes, and [19], [20] have learned about user behavior from software at runtime. Recently, [21] was able to extract events from `Eclipse` and have discovered, using a process mining tool, basic developers' workflows. Some statistics were computed based on the activities executed and artifacts edited. In [22], the authors have extracted development activities from non-instrumented applications and used machine learning algorithms to infer a set of basic development tasks, but no process mining techniques were used to discover any pattern of application usage. [23] used a semi-automatic approach for analyzing a large dataset of IDE interactions by using cluster analysis [23] to extract usage smells. More recently, [24] used process mining to gain knowledge on software in operation by analyzing the hierarchical events produced by application calls(eg: execution of methods within classes) at runtime. The studies mentioned above, extracted data from several different sources and have used a multitude of statistics methods, machine learning and process mining techniques. However, to the best of our knowledge, none of these works combine data from the IDE utilization with process mining methods with the aim of measuring individuals or teams efficiency. Even in the case of [21], where the approach is similar to ours, nothing was done related to conformance checking on the processes followed by developers, as there was no existing reference model to compare with. Our work introduces a valid

approach for this purpose, and bring a new perspective to software development analytics by filling this gap.

## III. Experiment

We analyzed several teams performing independently the same well-defined task on software quality assurance. To block additional confounding factors in our analysis, all teams had similar backgrounds and performed the same task upon the same software system. To provide authenticity, the task targeted a real-world (large) open-source Java system, the `Jasml (Java Assembling Language)`[1].

To understand what happened in each team, we mined the corresponding process model based on its events (process discovery phase). Then, we compared each discovered process with a reference model (process conformance checking phase), to assess the overall similarities and processes' quality.

### A. Research Questions

The following research questions emerged from our previously stated research goals:

RQ1) To what extent can process mining discover accurate models representing developers' behavior?

RQ2) Can we assess the efficiency of software development teams by using process mining techniques ?

RQ3) The assessment of teams' proficiency, performed by a process expert, is reflected in the quality of the produced models?

### B. Experimental Setup

*1) Subjects:* Subjects were finalists (3rd year) of a BSc degree on computer science at the ISCTE-IUL university, attending a compulsory software engineering course. By this time they had been trained across the same set of almost 30 courses and therefore had similar backgrounds. They worked in teams up to 4 members each and were requested to complete a code-smells detection assignment, aiming at identifying refactoring opportunities, using the `JDeodorant` tool[2]. This tool allowed the detection of four different types of code smells: `Long Method`, `God Class`, `Feature Envy` and `Type Checking` [25]. Once they have detected the occurrences of those code smells, they were required to apply `JDeodorant`'s automatic refactoring features to the critical ones.

*2) Data Collection Instrument:* The `Eclipse` IDE has an internal event bus accessed by the interface `IEventBroker`[3] which is instantiated once the application starts. It contains a publishing service to put data in the bus, whilst the subscriber service reads what's in that bus. This allows a subscriber to read all or part of the events being managed within the IDE. Using this feature we developed an `Eclipse` plugin[4] capable of listening to the actions developers were executing. Before the experiment, the plugin was installed on each subject work environment, and later, all received a unique *username/key* pair as credentials. This method was useful to unlock all the plugin features and allowed us to identify each subject and the corresponding team.

*3) Collected Data:* A sample event instance collected with our plugin is represented in listing 1 in `JSON` format. The field tags are self explanatory.

```
{
"team" : "T-01",
"session"  : "a5d63j-jdi3-ikd912",
"timestamp_begin" : "2018-05-07 16:53:52.144",
"timestamp_end" : "2018-05-07 16:54:04.468",
"fullname"  : "Ana Sample",
"username"  : "ana",
"workspacename"  : "Workspace1",
"projectname"  : "/jgrapht-core",
"filename"  : "/jgrapht-core/AncestorTest.java",
"extension"  : "java",
"categoryName": "Eclipse Editor",
"commandName": "File Editing",
"categoryID": "org.eclipse.ui.internal.EditorReference",
"commandID": "iscte.plugin.eclipse.commands.file.edit",
"platform_branch": "Eclipse Oxygen",
"platform_version": "4.7.3.M20180330-0640",
"java": "1.8.0_171-b11",
....
}
```

Listing 1: Sample Eclipse Event Instance

*4) Data Storage:* Collected data was stored locally in a CSV file. Whenever Internet connection was available, the same data was stored in the cloud[5]. This storage replication allowed offline and online collection. The final dataset, combining the two different sources, was then loaded into a MySQL database table where the username and event timestamps that formed the table's unique key were used for merging duplicated data. Figure 1 presents a schema of the data collection workflow.

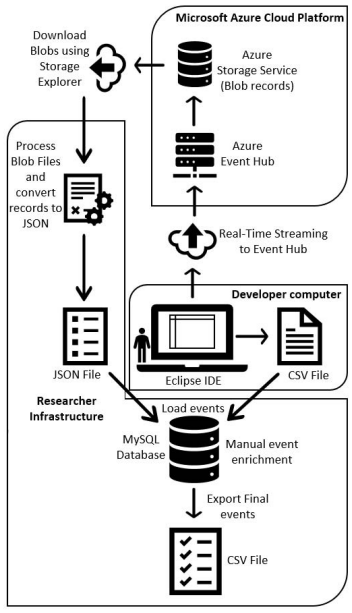



Fig. 1. Experiment Data Collection Workflow

[1]http://jasml.sourceforge.net/

[2]https://marketplace.eclipse.org/content/jdeodorant

[3]https://wiki.eclipse.org/Eclipse4/RCP/Event_Model

[4]https://github.com/jcaldeir/iscte-analytics-plugins-repository

[5]https://azure.microsoft.com/en-us/services/event-hubs/

*5) Data Preparation:* When the software quality task ended, all events stored in the database were converted to the IEEE eXtensible Event Stream (XES) standard format [26] and imported into ProM process mining tool[6]. The following event properties were mapped when converting to XES format:

- *team* was used as CaseID since we were interested to look into process instances of teams, not of individual programmers.
- Properties *categoryName* and *commandName* forming a hierarchical structure were used as the activity in the process.
- The *timestamp_begin* and *timestamp_end* were both used as activity timestamps.
- Other properties were used as a resource in the process.

*6) Data Demographics:* As previously mentioned, we only analyzed data collected on the same software system, to block confounding factors. The chosen system was `Jasml (Java Assembling Language)`[7].

The plugin collected two types of events: events within a project context**(PE)** and generic events**(GE)** at the `Eclipse` global context. The former summarizes events for which we have associated project and file names. This information expresses actions done by each developer in the project where `JDeodorant` features, such as, detecting a `God Class`, `Long Method`, `File Open`, `File Edit`, `Refactoring`, `Delete Resources`, were applied. The latter represents events captured from `Eclipse` command actions not associated with any project (e.g. `Update Eclipse Software`, `Install New Software`, `Open Eclipse View Task List`, etc).

We present their statistics in Table I. Project events should be seen as fundamental events for the task programmers were requested to execute, and, in a certain way represent the focus their are putting into that work. Generic events are seen as collateral actions not mandatory for the task in hand, but that programmers may need or want to execute to prepare their environment. These generic events somehow convey a lack of focus on the task developers were supposed to execute.

The `REFERENCE`(also identified as `REF.`) team, corresponds to the professor that proposed the task itself. Being the main expert, he executed it in one of the most efficient ways. The full dataset, that includes data on all teams with fine grained data that is not addressed in this paper, is publicly available.[8]

### C. Data Analysis

*1) Context:* Several approaches have been proposed to evaluate the quality of discovered process models. Software quality metrics were mapped to process metrics in [27]. Groups of metrics were also used in [28], [29] to evaluate several dimensions in a process model and, more recently, artifacts were created to support process quality evaluation and perform process variants comparisons [24], [30]. All of these fit within the well defined [15] and generally accepted four dimensions to assess the quality of a model: *fitness, precision, simplicity* and *generalization*.

*2) Process Discovery:* Several well known algorithms exist to discover process models, such as, the $\alpha$-algorithm, the heuristics, genetic and fuzzy miner. However, our need to discover and visualize the processes in multiple ways lead us to choose the `ProM's StateChart Workbench` plugin [24]. This plugin, besides supporting process model discovery using multiple hierarchies and classifiers, also allows to visualize the model as a Sequence Diagram and use notations such as Petri Nets and Process Trees. This plugin is particularly suitable for mining software logs, where an event structure is supposed to exist, but it also supports mining of other so-called generic logs.

Events collected from software in operation (e.g. Java programs) reveals the presence of a hierarchical structure, where methods reside within classes, and classes within packages [31]. The same applies to IDE usage actions, since identified menu options and executed commands belong to a specific category of command options built-in the `Eclipse` framework. Supported by this evidence, we used the Software log Hierarchical discovery method with a Structured Names heuristic, to discover the models based on the fact that the events were using a *category*|*command* structure (e.g. `Eclipse Editor|File Open`). Several perspectives can be used to discover and analyze a business process and the most commonly used are: `Control-Flow`, `Organizational`, `Social` and `Performance`. For the sake of space, we have just focused on the `Control-Flow` perspective in this paper. It defines an approach that consists in analyzing how each task/activity follows each other in an event log, and infer a possible model for the behavior captured in the observed process.

*3) Process Variant Comparison:* Our goal was also to compare the behaviour among the teams involved in the experiment against the "best practice" process, as performed by the expert, and identify the ones with less differences. For this purpose, we used the `Process Comparator` plugin [30], which is a tool that compares a collection of event logs, using a directed flow graph. It uses transition systems to model behavior and to highlight differences. Transition systems are annotated with measurements, and used to compare the behavior in the different variants. The annotations of each variant are compared using statistical significance tests, in order to detect relevant differences.

## IV. Results

Figure 2 presents team T-26 process variant, showing the code smells detection activities, and the correspondent statistics about the process followed to execute the requested task. It is clear, based on the different levels of blue in the activities performed, that they executed more often the activities related with the code smells detection and correction. We confirm this

[6]version 6.8, available at http://www.promtools.org

[7]http://jasml.sourceforge.net/

[8]doi:10.17632/8dmdwpgdy4.1

TABLE I
COLLECTED EVENTS STATISTICS

| Team | TM | UCC | UCA | UEA | PE (#/%) | GE (#/%) | TE (#) |
|---|---|---|---|---|---|---|---|
| T-43 | 4 | 10 | 38 | 39 | 790 / 85.13% | 138 / 14.87% | 928 |
| T-41 | 2 | 10 | 37 | 40 | 615 / 77.75% | 176 / 22.25% | 791 |
| T-02 | 3 | 12 | 41 | 24 | 552 / 74.80% | 186 / 25.20% | 738 |
| T-26 | 2 | 8 | 28 | 22 | 360 / 77.25% | 106 / 22.75% | 466 |
| T-23 | 1 | 9 | 23 | 22 | 276 / 93.24% | 20 / 6.76% | 296 |
| T-21 | 1 | 9 | 27 | 23 | 272 / 77.71% | 78 / 22.29% | 350 |
| T-24 | 1 | 8 | 26 | 13 | 181 / 89.60% | 21 / 10.40% | 202 |
| T-01 | 4 | 13 | 45 | 16 | 105 / 29.49% | 251 / 70.51% | 356 |
| REF. | 1 | 4 | 12 | 20 | 134 / 97.10% | 4 / 2.90% | 138 |

**TM** - Team members, **UCC** - Unique Command Categories, **UCA** - Unique Command Actions, **UEA** - Unique Edited Artifacts
**PE** - Project related events, **GE** - Generic Eclipse events, **TE** - Total events

by observing the `Eclipse Editor | File Editing` activity which was executed more than any other activity.

Globally, our attention went to the evaluation of the Simplicity (or Complexity) of the models discovered. Simplicity allude to the rule that the simplest model that can describe the behavior found in a log, is indeed the best model.

Software artifacts with higher cyclomatic complexity tend to be harder to maintain. It has been claimed that the same rationale is applicable to process models [32]. Based on this, we were looking for the teams with less complexity in their processes. As shown, teams T-26, T-24 and T-41 are the ones with less `Cyclomatic Complexity` (as represented by different levels of green), therefore closer to the complexity of the REFERENCE model. That is also reflected by the number of `Simple` and `Composite States`, and `Activities` discovered in each of those models. Team T-26 modelled behavior was also the one discovered with best precision (45%) among these 3 teams.

On the opposite pole (as represented by different levels of red) with an unique characterization, we have team T-01, with four members, which did not delivered the results of the requested task. Its proficiency was insufficient and careful review of the process revealed this team produced more generic events than project related events, as shown in Table I. From Figure 3 we can also learn this team used more unique command actions and respective categories than any other team, and that did not increase the number of edited files, as one would have expected. This leads us to think its members did not understand or follow the process at all, since many of their actions in the IDE apparently were not aligned with the required task. The high values of complexity, activities, number of transitions and composite states metrics observed in Table II complements this assumption.

We can, therefore, state the following: T-01 was an "expensive" team and the one that presented more risks from a project management perspective. When compared with other teams, this team had a similar process duration (see table IV) in executing the task, but did not deliver the expected outcomes at all. This team was not only non effective, but also showed major inefficiencies in whatever they tried to produce.

An interesting case to study deeper is team T-02 which had a good proficiency in the task, as seen in Table IV, but showed high levels of complexity in the model. This means we are dealing with a case where the team was effective, because they achieved the task with success, although without being efficient in the process. This is confirmed by the high number of different commands executed showed in Table I.

We also compared the behaviour between the 3 teams with less complex models against the reference model. The level of Control-Flow differences based on activity frequencies, as calculated with the Process Comparator plugin, is plotted in table III. Team T-24 was the one with less differences when compared with the reference model, followed very closely by T-26. Based on the complexity measurements, control-flow differences and team size, we advocate that T-26 had accomplished the task with the best overall efficiency and effectiveness. In fact, that is also reflected in the proficiency mark given by the professor (that acted as the task expert), as shown in table IV. This raises a set of other research questions, such as: can process mining be used to assess the proficiency of developers in general, or just for specific kinds of tasks?

TABLE II
MODELS DISCOVERED - METRICS SUMMARIZATION

| Team | F(%) | P(%) | A | HD | SS | CS | T | CC |
|---|---|---|---|---|---|---|---|---|
| T-43 | 85.8% | 39.9% | 37 | 2 | 93 | 12 | 130 | 35 |
| T-41 | 74.2% | 43.9% | 38 | 2 | 88 | 11 | 121 | 31 |
| T-02 | 81.6% | 33.8% | 47 | 2 | 109 | 11 | 159 | 48 |
| T-26 | 80.1% | 45.0% | 25 | 2 | 60 | 6 | 85 | 23 |
| T-23 | 79.7% | 32.3% | 31 | 2 | 104 | 16 | 141 | 35 |
| T-21 | 94.2% | 46.5% | 36 | 2 | 93 | 12 | 131 | 36 |
| T-24 | 94.4% | 35.9% | 30 | 2 | 74 | 8 | 103 | 27 |
| T-01 | 91.7% | 43.0% | 52 | 2 | 147 | 18 | 209 | 60 |
| REF. | 85.1% | 53.7% | 16 | 2 | 47 | 6 | 64 | 15 |

**F**-Fitness, **P**-Precision, **A**-Activities, **HD**-Hierarchy Depth,
**SS**-Simple States, **CS**-Composite States, **T**-Transitions,
**CC**-Cyclomatic Complexity

### A. Validity Threats

*1) Internal validity:* Since some teams worked in shared laboratories at the university campus, different team members may have used, in the same computer, the same user/key pair

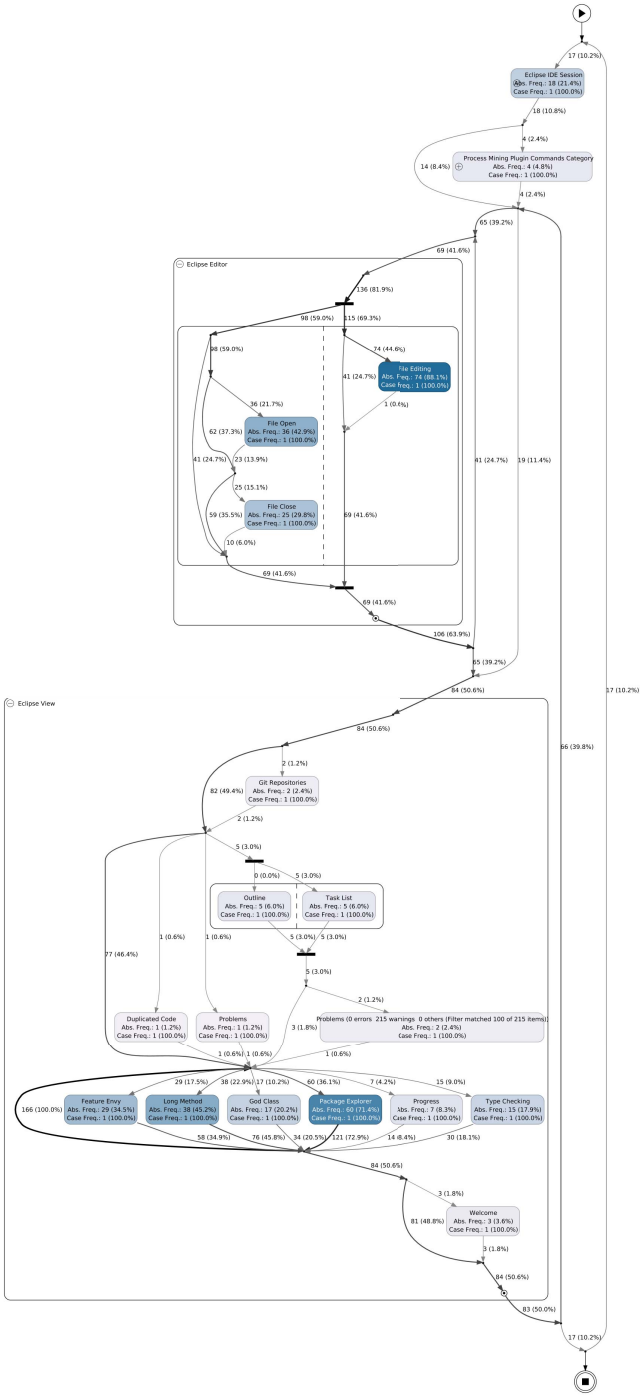



Fig. 2. Team T-26 Process Variant

to activate the collection plugin. This may be a source of non accuracy in collected data.

Some users have stopped the collection mechanism which makes it impossible to understand what they were doing during that period. We also found that a few teams have made a pause in the task, causing it to express more execution time than what

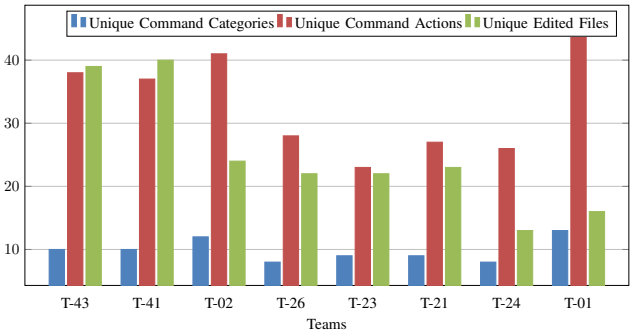



Fig. 3. Unique Categories,Commands and Edited Files Statistics

TABLE III
BEHAVIOR DIFFERENCES COMPARISON

| Ref. Log | Team | Control-Flow Differences(%) |
|---|---|---|
| REFERENCE | T-41 | 87.60 % |
| | T-26 | 85.11 % |
| | T-24 | 85.04 % |

TABLE IV
ASSIGNMENT DURATION

| Team | Proficiency | Process Duration |
|---|---|---|
| T-43 | 0 | 23h:49m |
| T-41 | 0.73 | 18d:3h |
| T-02 | 0.75 | 11d:22h |
| T-26 | 0.75 | 12d:16m |
| T-23 | 0.72 | 12d:13m |
| T-21 | 0.02 | 8d:12h |
| T-24 | 0.64 | 47m:14s |
| T-01 | 0 | 10d:7h |
| REFERENCE | – | 23m:05 |

was really needed. The mined processes reflects these times, but indeed, that was idle time. Nevertheless, other reasons may exist for these delays, and therefore, their model is in fact accurate, because it plots what really happened.

*2) External validity:* Since we wanted to block some factors such as the degree of previous experience (background) in the proposed process, and repeat the data collection process in a between groups design, to avoid the learning effects of paired designs, the only feasible solution was to use students as subjects, as referred in subsection III-B1. We cannot claim that these students are adequate surrogates for professional software developers.

## V. CONCLUSIONS AND FUTURE WORK

### A. Conclusions

*1) RQ1:* We can not underestimate the fact that software development IDEs provide the users with a vast number of commands and menus to execute from, as seen in II-C. Trying to model these, is indeed a challenge, and, most times, a spaghetti-like process is the result of a successful process discovery. However, from an event log containing user actions,

we were able to model teams' behavior with moderate-to-strong Fitness and Precision values and yet achieve readable models. We are however well aware that these values should be validated with more experiments and different data.

The need to answer RQ1 was vital to understand if we could indeed discover processes followed by different people, that may be using different tools, in different locations but contributing to the same final outcome or product. The importance of understanding and measure teams' dynamics has grown with the current business trends that lead to Global Software Engineering (GSE) and Global Software Development (GSD). This is one of the main challenges faced by GSE and GSD, as in those kinds of projects the usual monitoring techniques are obsolete [33].

*2) RQ2:* We were able to discover and reconstruct process models representing the efficiency of software development teams, where, in some cases, members were working individually, each with their own IDE setup configurations. We confirmed that process mining may play a fundamental role in assessing the efficiency of software development teams and in potentially contributing to keep them focused on their tasks by checking and enforcing compliance to the prescribed processes.

Every project manager wants to have in the projects he/she manages the most efficient and/or adequate resources. As this is expected to increase productivity in the development, measuring which teams or individuals are more efficient is a step further for better planning future software development projects.

*3) RQ3:* By assessing the way a task is executed and the proficiency achieved, as we did to answer RQ3, we were looking if there was any relation between those on the software development realm. This study can contribute to extend the discussion for the fact that the quality of a software product may well be dependent on the complexity of the processes followed.

In general, teams with less complexity in their models were among the most proficient in the task. This means that, they not only understood what was requested, but also had the maturity to deliver what was expected by following a simple process. They were not only effective, they were also efficient by being focused in the task.

On the contrary, teams with insufficient proficiency produced long and complex models or, in very short time, they created very fuzzy models with too many generic events. These teams were the ones where more risk aroused from a development project perspective due to their erratic behavior and uncertainty around the expected deliveries. Some of those teams did not perform very well and quality was impacted, and some others did not even deliver what was expected. In a real-world scenario, these teams would have been identified as the most expensive teams because their productivity was indeed very low.

This gives us some evidence that teams' proficiency can be inferred by analyzing mined process models representing their behavior. We don't see this as a coincidence, however, to sustain this evidence, we may need to replicate this experiment in other contexts and with a larger number of teams and developers.

No relevant performance or bottleneck patterns were identified in the processes, and the reason for this may be related with the type of task requested, which did not impose restrictions on times to work on any artifact, and/or the reduced schedule imposed on the task.

### B. Future Work

The current work can be expanded in breadth and in depth. In this paper we mainly explored the control-flow perspective, but others are worth exploring, such as the organizational and performance perspectives. Devising team dynamics based upon the identification of the artifacts impacted/touched by the developers can be one of the following paths to research further. This study also opens the opportunity for new research related with forensic analysis on software development processes, exploring a combined perspective of the quality of the artifacts produced and the underlying processes.

While unveiling the details of past process instances is important to understand what went wrong or unplanned, we should be able to react as soon as possible, that is, while the process is being executed, to enable just-in-time corrective actions. The IDE-based process mining architecture presented in this paper is forming the base of our **SPOTS** (**S**oftware **P**rocess **O**n-the-run **T**racking **S**ystem). This tool will provide near real-time software development process insights, at the individual or team level, such as in the Personal Software Process [34], or Team Software Process approaches [35], but in an automated fashion. According to [15], this kind of operational support is the most advanced form of process mining action.

We also plan to investigate how development process smells [36] may be used to assess software process drift management. Machine learning techniques are plausible candidates to automatically classify mined models (as good or bad process smells).

## ACKNOWLEDGMENT

The authors would like to thank to all ISCTE-IUL students involved in this research, as well as Prof. Vitor Basto Fernandes, who was in charge of the Software Engineering course where the experiment took place.

## REFERENCES

[1] P. Mohagheghi and M. Jorgensen, "What Contributes to the Success of IT Projects? Success Factors, Challenges and Lessons Learned from an Empirical Study of Software Projects in the Norwegian Public Sector," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 5 2017, pp. 371–373. [Online]. Available: http://ieeexplore.ieee.org/document/7965362/

[2] T. Chow and D.-B. Cao, "A survey study of critical success factors in agile software projects," *Journal of Systems and Software*, vol. 81, no. 6, pp. 961–971, 6 2008.

[3] M. Tanner and U. Von Willingh, "Factors leading to the success and failure of agile projects implemented in traditionally waterfall environments," in *Management, Knowledge and Learning*. Cape Town: University of Cape Town, South Africa, 2014, pp. 693–700. [Online]. Available: http://www.toknowpress.net/ISBN/978-961-6914-09-3/papers/ML14-618.pdf

[4] A. Aldahmash, A. M. Gravell, and Y. Howard, "A Review on the Critical Success Factors of Agile Software Development," in *European Conference on Software Process Improvement*. Springer, Cham, 2017, pp. 504–512.

[5] M. Borges Ribeiro, V. Diniz Duarte, E. Gomes Salgado, and C. Vieira Castro, "Prioritization of Critical Success Factors In The Process of Software Development," *IEEE Latin America Transactions*, vol. 15, no. 1, pp. 137–144, 1 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7827917/

[6] K. E. Emam and A. G. Koru, "A Replicated Survey of IT Software Project Failures," *IEEE Software*, vol. 25, no. 5, pp. 84–90, 9 2008. [Online]. Available: http://ieeexplore.ieee.org/document/4602680/

[7] P. A. McQuaid, "Software disasters—understanding the past, to improve the future," *Journal of Software: Evolution and Process*, vol. 24, no. 5, pp. 459–470, 2012.

[8] IEEE Computer Society, *SWEBOK V3.0*. IEEE Computer Society, 2014, no. V3.0. [Online]. Available: http://www4.ncsu.edu/ tjmenzie/cs510/pdf/SWEBOKv3.pdf www.swebok.org

[9] M. Niazi, S. Mahmood, M. Alshayeb, M. R. Riaz, K. Faisal, N. Cerpa, S. U. Khan, and I. Richardson, "Challenges of project management in global software development: A client-vendor analysis," *Information and Software Technology*, vol. 80, no. C, pp. 1–19, 12 2016. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0950584916301227

[10] A. M. Lemos, C. C. Sabino, R. M. F. Lima, and a. L. Oliveira, "Conformance Checking of Software Development Processes Through Process Mining," *Seke*, 2011.

[11] J. Caldeira and F. Brito E Abreu, "Software development process mining: Discovery, conformance checking and enhancement," in *Proceedings - 2016 10th International Conference on the Quality of Information and Communications Technology, QUATIC 2016*. IEEE, 9 2017, pp. 254–259. [Online]. Available: http://ieeexplore.ieee.org/document/7814558/

[12] W. Poncin, A. Serebrenik, and M. V. D. Brand, "Process Mining Software Repositories," *2011 15th European Conference on Software Maintenance and Reengineering*, pp. 5–14, 2011.

[13] W. Van Der Aalst, A. Adriansyah, A. K. A. De Medeiros, F. Arcieri, T. Baier, T. Blickle, J. C. Bose, P. Van Den Brand, R. Brandtjen, J. Buijs, A. Burattin, J. Carmona, M. Castellanos, J. Claes, J. Cook, N. Costantini, F. Curbera, E. Damiani, M. De Leoni, P. Delias, B. F. Van Dongen, M. Dumas, S. Dustdar, D. Fahland, D. R. Ferreira, W. Gaaloul, F. Van Geffen, S. Goel, C. Günther, A. Guzzo, P. Harmon, A. Ter Hofstede, J. Hoogland, J. E. Ingvaldsen, K. Kato, R. Kuhn, A. Kumar, M. La Rosa, F. Maggi, D. Malerba, R. S. Mans, A. Manuel, M. McCreesh, P. Mello, J. Mendling, M. Montali, H. R. Motahari-Nezhad, M. Zur Muehlen, J. Munoz-Gama, L. Pontieri, J. Ribeiro, A. Rozinat, H. Seguel Pérez, R. Seguel Pérez, M. Sepúlveda, J. Sinur, P. Soffer, M. Song, A. Sperduti, G. Stilo, C. Stoel, K. Swenson, M. Talamo, W. Tan, C. Turner, J. Vanthienen, G. Varvaressos, E. Verbeek, M. Verdonk, R. Vigo, J. Wang, B. Weber, M. Weidlich, T. Weijters, L. Wen, M. Westergaard, and M. Wynn, "Process mining manifesto," *Lecture Notes in Business Information Processing*, vol. 99 LNBIP, pp. 169–194, 2012.

[14] M. L. Van Eck, X. Lu, S. J. J. Leemans, and W. M. P. Van Der Aalst, "PM 2 : a Process Mining Project Methodology," in *International Conference on Advanced Information Systems Engineering CAiSE 2015: Advanced Information Systems Engineering*. Springer, Cham, 2015, pp. 297–313.

[15] W. van der Aalst, *Process Mining : Data Science in Action*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016.

[16] A. Fuggetta, E. D. Nitto, and P. Milano, "Software Process," *Proceedings of the on Future of Software Engineering*, pp. 1–12, 2014.

[17] A. Meidan, J. A. García-García, I. Ramos, and M. J. Escalona, "Measuring Software Process," *ACM Computing Surveys*, vol. 51, no. 3, pp. 1–32, 6 2018. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3212709.3186888

[18] A. R. C. Maita, L. C. Martins, C. R. López Paz, L. Rafferty, P. C. K. Hung, S. M. Peres, and M. Fantinato, "A systematic mapping study of process mining," *Enterprise Information Systems*, vol. 12, no. 5, pp. 505–549, 5 2018. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/17517575.2017.1402371

[19] V. A. Rubin, I. Lomazova, and W. M. P. v. d. Aalst, "Agile development with software process mining," *Proceedings of the 2014 International Conference on Software and System Process - ICSSP 2014*, pp. 70–74, 2014.

[20] V. A. Rubin, A. A. Mitsyuk, I. A. Lomazova, and W. M. P. van der Aalst, "Process mining can be applied to software too!" *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '14*, pp. 1–8, 2014.

[21] C. . Ioannou, A. . Burattin, and B. Weber, "Mining Developers' Workflows from IDE Usage," *Lecture Notes in Business Information Processing*, vol. 316, pp. 167–179, 2018.

[22] L. Bao, Z. Xing, X. Xia, D. Lo, and A. E. Hassan, "Inference of development activities from interaction with uninstrumented applications," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1313–1351, 6 2018. [Online]. Available: http://link.springer.com/10.1007/s10664-017-9547-8

[23] K. Damevski, D. C. Shepherd, J. Schneider, and L. Pollock, "Mining Sequences of Developer Interactions in Visual Studio for Usage Smells," *IEEE Transactions on Software Engineering*, vol. 43, no. 4, pp. 359–371, 4 2017.

[24] M. Leemans, W. M. P. van der Aalst, and M. G. J. van den Brand, "The Statechart Workbench: Enabling scalable software event log analysis using process mining," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 3 2018, pp. 502–506. [Online]. Available: http://ieeexplore.ieee.org/document/8330248/

[25] K. Beck, M. Fowler, J. Brant, W. Opdyke, and D. Roberts, "Bad Smells in Code," in *... Improving the design of existing code*, 1999.

[26] C. Günther and E. Verbeek, "XES standard definition," *Fluxicon Process Laboratories*, 2014.

[27] I. Vanderfeesten, J. Cardoso, J. Mendling, H. A. Reijers, and W. Van Der Aalst, "Quality Metrics for Business Process Models," Technische Universiteit Eindhoven, Tech. Rep., 2007. [Online]. Available: http://wwwis.win.tue.nl/ wvdaalst/publications/p364.pdf

[28] A. Rozinat, A. de Medeiros, C. Günther, A. Weijters, and W. van der Aalst, "Towards an evaluation framework for process mining algorithms," *Beta, Research School for Operations Management and Logistics.*, pp. 1–20, 2007. [Online]. Available: http://www.processmining.org.

[29] A. Rozinat, M. Veloso, and W. M. P. van der Aalst, "Evaluating the Quality of Discovered Process Models," *Information Systems Journal*, vol. 16, no. Section 2, pp. 1–8, 2008.

[30] A. Bolt, M. de Leoni, and W. M. van der Aalst, "Process variant comparison: Using event logs to detect differences in behavior and business rules," *Information Systems*, vol. 74, pp. 53–66, 5 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306437916305257

[31] M. Leemans, W. M. P. van der Aalst, and M. G. J. van den Brand, "Recursion aware modeling and discovery for hierarchical software event log analysis," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2018, pp. 185–196.

[32] J. Cardoso, J. Mendling, G. Neumann, and H. A. Reijers, "A discourse on complexity of process models," in *International Conference on Business Process Management*. Springer, 2006, pp. 117–128.

[33] F. Jurado and P. Rodriguez, "Sentiment Analysis in monitoring software development processes: An exploratory case study on GitHub's project issues," *Journal of Systems and Software*, vol. 104, pp. 82–89, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121215000485

[34] W. S. Humphrey, "Personal Software Process (PSP)," in *Encyclopedia of Software Engineering*, 2nd ed., J. J. Marciniak, Ed. New York, NY, USA: John Wiley & Sons, 2002, p. 1584.

[35] ——, "Team Software Process (TSP)," *Encyclopedia of Software Engineering*, 2002.

[36] B. Weber, M. Reichert, J. Mendling, and H. A. Reijers, "Refactoring large process model repositories," *Computers in Industry*, vol. 62, no. 5, pp. 467–486, 2011.

# Improving Alignment Computation using Model-based Preprocessing

Alifah Syamsiyah
Eindhoven University of Technology
The Netherlands
Email: a.syamsiyah@tue.nl

Boudewijn F. van Dongen
Eindhoven University of Technology
The Netherlands
Email: b.f.v.dongen@tue.nl

***Abstract*—Alignments are a fundamental approach in conformance checking to provide an explicit relation between traces of events observed in an event log and execution sequences of process models. They are robust against intricacies in process models such as duplicate labels and invisible transitions, but at the same time computing them is a time consuming task. In this paper, we argue that precomputed rules may be leveraged to improve on the time needed to compute alignments. To this end, we utilize both structural and behavioral properties of process models to derive rules and we compare events against these rules. A violation in one of the rules indicates a problem in the event. Before alignments are computed, we mark the problematic events as so-called splitpoints. We evaluated this approach on real-life logs as well as benchmarking logs, and the results show that the proposed approach is faster than existing alignment approaches.**

## I. Introduction

Reality often deviates from what has been designed and business process executions are no exception. That is why conformance checking has become a significant field in process mining [4]. In conformance checking, the goal is to relate reality captured in event logs with designed (or discovered) process models. The state-of-the-art of conformance checking uses alignments to exactly capture this relation.

Alignments are robust against intricacies in process models such as duplicate labels and invisible transitions. Moreover, they are precise and are the basis for various metrics to determine the adequacy of a process model in describing a log. However, computing globally optimal alignments is a complex task and challenging from a computational point of view.

Alignments transform a conformance checking problem into a shortest path problem of a *synchronous product net*. This net covers both execution sequences of a trace and a model and finding an optimal alignment is equivalent to finding a shortest path between an initial and a final marking in a synchronous product net. The runtime complexity of finding alignments is exponential in the size of a synchronous product in the worst case scenario.

Alignments harness the A$^\star$ algorithm as the shortest path algorithm. A cost function is used to associate costs to specific transitions in the synchronous product which correspond to deviations. A$^\star$ depends on the quality of a heuristic function $h$ which *underestimates* the total remaining cost. If $h$ is of poor quality, A$^\star$ visits almost the entire space thus leading to extremely high computation time. Traditionally, the $h$ value is calculated using Petri net theory and it is based on a solution vector of a marking equation minimizing the cost function. In general, this solution vector does not necessarily correspond to a realizable firing sequence, which leads to A$^\star$ searching through large search spaces.

To tackle this issue, an approach called incremental alignments was introduced in [20]. This approach splits the marking equation to guarantee that a transition is enabled before firing, i.e. there are enough tokens present in the input place of the transition. By splitting the marking equation in the heuristic function, it becomes more accurate and therefore the A$^\star$ search becomes faster. Unfortunately, in [20], it is not known in advance where to split the marking equation which causes a loss of time due to frequent restarts of the search.

In this paper, we present a technique to precompute the splitpoints using properties of the model. As these splitpoints are model-based, they can be precomputed and stored next to event data in a database thus reducing the time needed when doing conformance checking. We utilize rules based on places (i.e. structural properties) and/or behavioral profiles (i.e. behavioral properties) to identify splitpoints. Technically, events are pre-checked against these rules in a linear pass over the log. If at any point one of the rules is violated, the index is set as a splitpoint where the marking equation is extended.

To conclude, the focus of this paper is *to improve alignment computation through predefined splitpoints based on properties of process models*. The remainder of this paper is structured as follows. In Section 2 we elaborate the related work and in Section 3 we introduce some important terminology. In Section 4, we explain our contribution in detail and in Section 5 we introduce the implementation of the experiments in Section 6. We conclude the paper in Section 7.

## II. Related Work

Alignments were first introduced in [1]. Although this seminal work put a solid basis for analyzing the conformity between process models and event logs, computing alignments is computationally expensive. For more details on alignments and how to compute them, we refer to [4] and references there.

The work in [17] proposed an iterative way to speed up alignment computation. Using a marking equation, it computes a prefix alignment of limited size $s$ that avoid passing through

978-1-7281-0919-0/19/$31.00 ©2019 IEEE
DOI 10.1109/ICPM.2019.00021

negative markings. After the prefix is constructed, the search continues with the same length $s$ until the final marking is reached. The drawbacks of this strategy are considerable growth of ILP formulation and the number of steps to reach the final marking needs to be known in advance. The work in [12] enables alignments in the large by decomposed conformance checking. This work presented a novel decomposition technique called Single-Entry Single-Exit that partitions large process models and event logs into smaller parts that can be analyzed independently. All these techniques are designed for efficiently computing alignments, however the alignments are not guaranteed to be optimal.

To overcome the issue with alignments not being optimal, recomposing conformance [7] checking builds on [12] by iteratively changing the decomposition of the problem whenever the result is sub-optimal. This work guarantees optimality, but for each sub-problem it uses a classic alignment technique. Therefore, the work presented in this paper seamlessly integrates with this framework. This also holds for the optimizations presented in [21]. This work optimizes the A$^\star$ search algorithm using 2$^{\text{nd}}$-order sorting criteria within an internal priority queue and some parametrization which utilizes Petri net theory. Despite the set of optimizations on the original A$^\star$ algorithm, however, very complicated models still require to visit almost the entire A$^\star$ search space.

Several other techniques exist to compute alignments. The work in [2] maximizes synchronization between an event log and a model and in [6] alignments are computed for safe Petri nets using show-floor planner software. For safe, acyclic models, constraint satisfaction [8] can be used. When not using a model, but rather its reachability graph as input, automata matching [13] can be applied. The models in this paper are safe. However, they are cyclic and, due to parallelism, the reachability graphs cannot be computed in reasonable time.

The most recent approach proposed in [20] utilized extended marking equation to provide a more accurate underestimation of the remaining cost. The initial solution vector returned by the heuristic function is sequentialized maximally. If this cannot be done further, specific events are marked as splitpoints and the search restarts with a different heuristic function. In this paper, we build on the ideas outlined in [20] and we precompute splitpoints using properties of the model rather than letting the A$^\star$ search find them incrementally.

Finally, the work in [22] considers conformance checking from behavioral perspective. Rather than grounding the problem into state-based techniques, it leverages behavioral abstraction to measure the compliance of logs versus models. We harness the behavioral profiles mentioned in this work as part of our rules.

## III. Preliminaries

### A. Event Log

Let $\mathcal{U}_\mathcal{A}$ be a universe of activities. An event log $\mathcal{L}^E$ is a multiset of sequences over $\mathcal{U}_\mathcal{A}$, i.e. $\mathcal{L}^E \in \mathcal{B}({\mathcal{U}_\mathcal{A}}^*)$. Furthermore, a trace is a sequence of activities, i.e. $\sigma \in \mathcal{L}^E$. We denote the i$^{\text{th}}$ element of $\sigma$ with $\sigma[i]$ and the subsequence from the $i^{th}$ to the $j^{th}$ element as $\sigma[i..j]$ where $\sigma[i..j] = \langle\rangle$ if $j < i$.

### B. Petri Nets

Petri nets are a process modeling language in the form of bipartite graphs that consist of places and transitions. In this paper, we use the notion of labeled Petri nets where each transition is labeled with an activity name or a silent label $\tau$.

**Definition 1** (Labeled Petri Nets)**.** *Let $\mathcal{P}$ be a set of places, $\mathcal{T}$ be a set of transitions, and $\mathcal{F} \subseteq (\mathcal{T} \times \mathcal{P}) \cup (\mathcal{P} \times \mathcal{T})$ be a flow relation. Moreover, let $\lambda : \mathcal{T} \to \mathcal{U}_\mathcal{A} \cup \{\tau\}$ denote a labeling function. A Petri net is quadruple $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, \lambda)$.*

A marking $m$ of $\mathcal{N}$ is a multiset of places, i.e. $m \in \mathcal{B}(\mathcal{P})$. We denote the initial and final marking of $\mathcal{N}$ as $m_i$ and $m_f$, respectively. In this paper, we use Petri nets with a clear initial and final marking. We denote the language of the Petri net as $\mathcal{L}^\mathcal{N} \subseteq \mathcal{T}^*$ and we assume this is the set of sequences of transitions which are enabled in the initial marking $m_i$ and terminate in the final marking $m_f$.

An example of a Petri net is shown in Figure 1. Here, a model with 5 places (circles) and 5 transitions (boxes) is shown. Each transition is labeled with an activity from the set $\{a, b, c, d, e\}$.

### C. Alignments

An alignment is a way of relating events in a trace to transitions in a Petri net, in such a way that the transitions form a full execution of the model from initial to final marking. Typically, when using alignments, an optimal alignment is sought, i.e. one that minimizes a certain cost function that associates costs to events not represented by the firing of a corresponding transition, or transition firings which do not correspond to events. It is beyond the scope of this paper to formally define alignments and we refer to [4] for an in depth discussion on them and on how to compute them.

### D. Computing Alignments

Traditionally, finding optimal alignments is done using A$^\star$ on the state space of a so-called synchronous product of the Petri net and the trace. The synchronous product for the model of Figure 1 and the trace $\langle a, e, d\rangle$ is shown in Figure 2. This model is again a Petri net. The top part are the transitions of the original model (i.e. model moves) and the bottom part are the events in the log represented as labeled transitions (i.e. log moves). In the middle, three synchronous transitions are shown that combine transitions with identical labels from the top and bottom model (i.e. synchronous moves).

The problem of finding an optimal alignment is identical to the problem of finding a firing sequence in the synchronous product while minimizing the number of model- or log-moves.

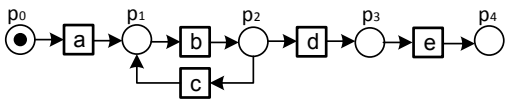



Fig. 1. A simple Petri net.

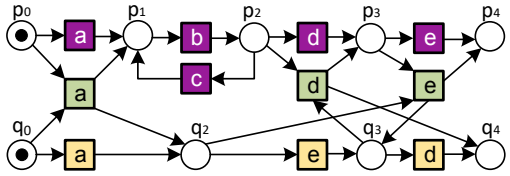



Fig. 2. The synchronous product for the model in Figure 1 and the trace $\langle a, e, d \rangle$.

The A$^\star$ technique used in [1], [20] relies on a heuristic function $h$ to underestimate the remaining costs to reach the final marking in the model while explaining all remaining events.

In [20], a technique was presented to compute alignments using an incremental approach. This approach works by iteratively identifying events which are known to be problematic. In the underlying A$^\star$ search technique, these events are identified as splitpoints, i.e. the heuristic function is enhanced to ensure these events are explained properly. Without going into too much technical detail, it suffices to say that the technique starts with heuristic function $h_\emptyset$. Whenever an event $\sigma[i]$ at index $i$ of the trace $\sigma$ is found to be problematic, this index is added to the heuristic, i.e. we get $h_{\{i\}}$ and the search starts again. Index $i$ is then referred to as a splitpoint.

For the example in Figure 2, the heuristic function $h_\emptyset$ is unable to identify the swapped activities right from the start. In the first iteration, the function is changed to $h_{\{3\}}$ after which the heuristic function will correctly identify the swapped activities and will be able to quickly identify an optimal alignment with three deviations, namely:

| log trace | $a$ | $\gg$ | $e$ | $d$ | $\gg$ |
|---|---|---|---|---|---|
| execution sequence | $a$ | $b$ | $\gg$ | $d$ | $e$ |

While this technique is proven effective for certain cases, identifying splitpoints requires a partial search through the state space of the synchronous product and each additional splitpoint requires the technique to search through a larger and larger part thereof. Therefore, in the following section, we show two approaches to precompute splitpoints.

## IV. Alignment with Predefined Splitpoints based on Rule Checking

Different to the incremental alignments which let the A$^\star$ algorithm split during the search, we give A$^\star$ hints of where to split before the alignment computation starts using *rule checking*.

The idea behind this is simple. A Petri net is essentially a set of rules which have to hold when executing the model, e.g. it cannot consume more tokens from a place than have been produced in this place. Such rules, which are provided at the level of transitions, can be translated to rules on the level of activities. As each event in a trace also refers to an activity, we can decide for which events these rules are violated. If an event violates one of the rules, the event index is stored as a splitpoint.

A violation of these rules intuitively implies that, in order to explain the event using the model, additional transitions need to be fired (which in alignment terms would be "move on models") or the event should be flagged as a deviation (a "move on log"). By adding the index of an event violating a rule to the heuristic function, we basically force the heuristic in the A$^\star$ search algorithm to explain the event properly.

There are two techniques to define rules of a process model: place-based rules (Section IV-A) which use the places of the Petri net, and behavioral-profile based rules (Section IV-B) which use more long-term relations between activities.

### A. Place-based Rules

Given a Petri net $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, \lambda)$, place-based rules are a set of constraints reflecting the structural properties of $\mathcal{N}$. In order to enable a transition, there must be enough tokens available in the corresponding places. Therefore, for each place $p \in \mathcal{P}$, at any point in a firing sequence of that Petri net, the number of executions of input transitions must be greater or equal to the number of executions of output transitions.

**Definition 2** (Place-based Rules ($\rho_\mathcal{N}$))**.** *Let $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, \lambda)$ be a Petri net with an initial marking $m_i$ and a final marking $m_f$. For all $p \in \mathcal{P}$, $\bullet p$ is the set of input transitions of p, i.e. $\bullet p = \{t | (t,p) \in \mathcal{F}\}$, and $p\bullet$ is the set of output transitions of p, i.e. $p\bullet = \{t | (p,t) \in \mathcal{F}\}$.*

*For each place $p \in \mathcal{P}$, we define the place-based rule $\rho_p : \mathcal{U_A}^* \to \{\top, \bot\}$ such that for $\sigma \in \mathcal{U_A}^*$ holds: $\rho_p(\sigma) = m_i(p) + \#_{\substack{t \in \bullet p \\ 1 \le i \le |\sigma|}} \sigma[i] = \lambda(t) \geq \#_{\substack{t \in p\bullet \\ 1 \le i \le |\sigma|}} \sigma[i] = \lambda(t)$.*

*i.e. $\rho_p$ is an inequation over activities in $\mathcal{U_A}$. Note that we use $\top$ and $\bot$ to denote true and false respectively. We use $\rho_\mathcal{N}$ to denote the full set of rules, i.e. $\rho_\mathcal{N} = \{\rho_p \mid p \in \mathcal{P}\}$.*

Consider a simple process model $\mathcal{N}$ in Figure 1. According to Definition 2, the place-based rules of $\mathcal{N}$ are as follows. Place $p_0$ is a source place containing one token, hence the rule is $\rho_{p_0}(\sigma) = 1 \geq \#_{1 \le i \le |\sigma|} \sigma[i] = a$, or using shorthand notation: $1 \geq a$. Place $p_1$ has both $a$ and $c$ as the input transitions and $b$ as the output transition, hence the rule is $a + c \geq b$. Similarly, the rules of $p_2$ and $p_3$ are $b \geq c + d$ and $d \geq e$, respectively. Meanwhile, place $p_4$ induces the trivial rule $e \geq 0$ since it is a sink place which does not have any output transitions.

Using the function $\rho_p$, a splitpoint is defined at each index $i$ in a trace if the result of the rule is true up to index $i-1$ and then becomes false.

**Definition 3** (Splitpoints based on $\rho_\mathcal{N}$ ($sp_\sigma$))**.** *Let $\mathcal{L}^E \in \mathcal{B}(\mathcal{U_A}^*)$ be an event log and $\sigma$ is a trace in $\mathcal{L}^E$. Let $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, \lambda)$ be a Petri net.*

TABLE I
Checking a trace $\sigma = \langle a, c, b, c, b, d, e \rangle$ based on place-based rules leads to the splitpoints $sp_\sigma = \{2, 4, 6\}$.

| | $\sigma =$ | | | | | | |
|---|---|---|---|---|---|---|---|
| $\rho_\mathcal{N}$ | $\langle a$ | $c$ | $b$ | $c$ | $b$ | $d$ | $e\rangle$ |
| $1 \geq a$ | $\top$ | $\top$ | $\top$ | $\top$ | $\top$ | $\top$ | $\top$ |
| $a + c \geq b$ | $\top$ | $\top$ | $\top$ | $\top$ | $\top$ | $\top$ | $\top$ |
| $b \geq c + d$ | $\top$ | $\bot$ $_{0 \not\geq 1+0}$ | $\top$ | $\bot$ $_{1 \not\geq 2+0}$ | $\top$ | $\bot$ $_{2 \not\geq 2+1}$ | $\bot$ $_{2 \not\geq 2+1}$ |
| $d \geq e$ | $\top$ | $\top$ | $\top$ | $\top$ | $\top$ | $\top$ | $\top$ |
| $e \geq 0$ | $\top$ | $\top$ | $\top$ | $\top$ | $\top$ | $\top$ | $\top$ |
| splitpoints: | | 2 | | 4 | | 6 | |

*The set of splitpoints* $\text{sp}_\sigma$ *for* $\sigma$ *is defined as follows. For* $1 \leq i \leq |\sigma|$ *holds that* $i \in \text{sp}_\sigma$ *if and only if there exists a place* $p \in \mathcal{P}$*, such that* $\rho_p(\sigma[1..i-1]) = \top$ *and* $\rho_p(\sigma[1..i]) = \bot$.

Consider again a process model $\mathcal{N}$ depicted in Figure 1. Let $\sigma = \langle a, c, b, c, b, d, e \rangle$ be a trace to be aligned with $\mathcal{N}$. The place-based rules of $\mathcal{N}$ ($\rho_\mathcal{N}$) are displayed on the first column of Table I. Based on this set of rules, we replay $\sigma$ and obtain the following splitpoints $sp_\sigma = \{2, 4, 6\}$. This means the 2nd, the 4th, and the 6th event of $\sigma$ violate some rules in $\rho_\mathcal{N}$. As shown in Table I, all these events violate the third rule as the number of occurrences of $b$ in the trace is less than the number of occurrences of $c$ and $d$. The 7th event also violates the third rule. However, it does not introduce a splitpoint as the rule is already violated before.

The example above uses a simple Petri net without any duplicate labels and silent transitions. Because of the one-to-one correspondence between transitions and activities, the places, which impose restrictions on the number of firings of transitions, directly translate to rules on the level of activities. In the reminder of this section, we show how to deal with duplicate activities and so-called $\tau$-labeled transitions.

*1) Handling Duplicate Labels:* Consider a Petri net with duplicate labels in Figure 3. According to Definition 2, the place-based rules of the net are $\rho_\mathcal{N} = \{1 \geq a, a \geq b, b \geq a, a \geq c, c \geq 0\}$. Suppose that a trace $\sigma = \langle a, b, a, c \rangle$ is checked against $\rho_\mathcal{N}$. This trace perfectly fits with the net, however, the rule checking produces splitpoints $sp_\sigma = \{1, 3\}$. Based on $\rho_\mathcal{N}$, the 1st event violates the constraint $b \geq a$ while the 3rd violates $1 \geq a$.

The cause of this violation is the fact that there are two transitions labeled $a$ in the model and we cannot distinguish two different $a$'s within a trace. Since it is impossible to make any assumptions on which event belongs to which transitions, we aggregate the rules pertaining to activity $a$ by adding them.

Technically, first we rewrite the rules to ensure unique labels, i.e. we change activity $a$ into $a_1$ and $a_2$, hence the rules become $\rho_\mathcal{N}{}' = \{1 \geq a_1, a_1 \geq b, b \geq a_2, a_2 \geq c, c \geq 0\}$. If we add the 1st and 3rd constraints in $\rho_\mathcal{N}{}'$, we obtain $1 + b \geq a_1 + a_2$, i.e. the initial token and the tokens produced by firing $b$ should suffice, at any point in the trace, to fire any of the transitions referring to $a$. It is clear that the total occurrences of $a$ in any valid execution sequence equals to sum over the duplicate labels of $a$, i.e. $a = a_1 + a_2$. Therefore, $1 + b \geq a_1 + a_2$ is equivalent to $1 + b \geq a$. With the same technique, we obtain $a \geq b + c$ from adding $a_1 \geq b$ and $a_2 \geq c$. Finally, the rules become $\rho_\mathcal{N}{}' = \{1 + b \geq a, a \geq b + c, c \geq 0\}$. If we check $\sigma$ against $\rho_\mathcal{N}{}'$, all events in $\sigma$ are satisfied with $\rho_\mathcal{N}{}'$ as we expected.

Note that aggregating rules in this manner can always be done as these are sets of inequalities and one can always add arbitrary transitions to the left-hand side as well as remove arbitrary transitions from the right-hand side, thereby weakening the constraint.

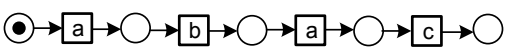



Fig. 3. A Petri net with duplicate labels.

*2) Handling Silent Transitions:* Besides duplicate labels, we also frequently encounter silent transitions in process models. A silent transitions (or $\tau$-labeled transition) does not correspond to any activity label in a trace. Therefore, if it appears in a rule, we cannot decide how many times such a transition has fired at any point in the trace and we need to eliminate $\tau$ from rules. Let $p$ be the place in which $\tau$ is connected to. There are two scenarios, namely $\tau$ is either an output or input transition of $p$. In the first case, we ignore $\tau$ by deleting it from the rule. A deletion does not affect any behavior except that it makes the constraint less restrictive. Consider a Petri net in Figure 4i. The rule of $p$ is $a \geq b + \tau$ which is transformed to $a \geq b$.

In the second case, we cannot remove $\tau$ as before because the inequality may no longer hold after the removal (note that from $\tau + x \geq y$ we cannot conclude that $x \geq y$). Our approach is to backtrack and find the closest place $p'$ which has a non silent input transition $t$. We then add $p'$ with $p$ and combine their input and output transitions as in duplicate labels. As illustrated in Figure 4ii, we combine $p$ and $p'$ which yields $a + c + d \geq b + e$ after removing $\tau$ in both sides.

However, there is a possibility in which not only one place connected to $\tau$, i.e. $\tau$ serves as an AND-join. If the connected places are $p'$ and $p''$, there will be two place combinations, namely $\{p', p\}$ and $\{p'', p\}$. This results to two new rules, as illustrated in Figure 4iii. By iterating over the model in a forward and/or backward direction, all $\tau$'s can be eliminated, except if a $\tau$ loop is found. In that case the rule is simply removed from the set.

Note that dealing with duplicate labels and dealing with silent transitions are both non-trivial tasks for more complex net structures[1]. However, for the net structures typically encountered in practice the intuition provided above is sufficient, also from a computational point of view.

A second technique to pre-compute rules using the Petri net as input is by using so-called behavioral profiles.

### B. Behavioral Profile-based Rules

To define behavioral profile-based rules, we first define eventually follows relations.

**Definition 4** (Eventually Follows Relations)**.** *Let* $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, \lambda)$ *be a Petri net with an initial marking* $m_i$ *and a final marking* $m_f$*. For all* $a, b, \in \mathcal{U}_\mathcal{A}$*, a is eventually followed by b, i.e.* $a \ggg b$*, if and only if* $\exists_{\gamma \in \mathcal{L}^\mathcal{N}} \exists_{1 \leq j < i \leq |\gamma|} \lambda(\gamma[j]) = a \wedge \lambda(\gamma[i]) = b$.

Based on these eventually follows relations we now define the behavioral profile-based rules.

**Definition 5** (Behavioral Profile)**.** *Let* $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, \lambda)$ *be a Petri net and let* $\mathcal{A}$ *be a set of activity labels such that* $\mathcal{A} = \{\lambda(t) \neq \tau \mid t \in \mathcal{T}\}$*. For all* $a, b \in \mathcal{A}$*:*

1) *a is in strict order relation with b, i.e.* $a \rightsquigarrow b$ *if and only if* $a \ggg b \ \wedge \ b \not\ggg a$.

[1]This problem equals the variable elimination problem for systems of linear inequations which has a complexity that is worst case double exponential in the number of variables that need to be reduced, i.e. the set $\mathcal{T}$.

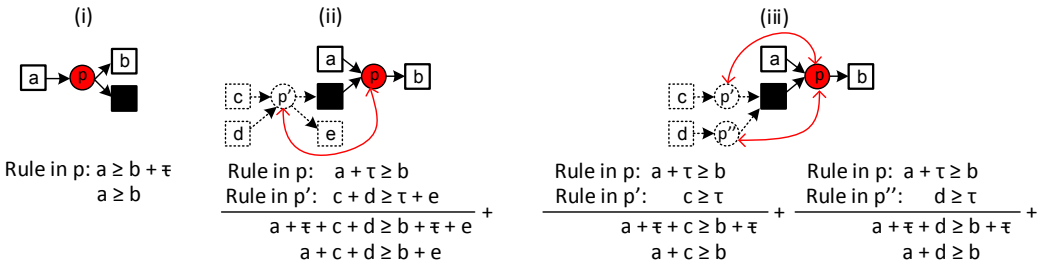



Fig. 4. Different scenarios in handling silent transitions.

2) *$a$ is in reverse strict order relation with $b$, i.e. $a \nleftarrow b$ if and only if $b \rightsquigarrow a$.*
3) *$a$ is in exclusiveness relation with $b$, i.e. $a + b$ if and only if $a \not\gg b \ \wedge \ b \not\gg a$.*
4) *$a$ is in interleaving order relation with $b$, i.e. $a||b$ if and only if $a \ggg b \ \wedge \ b \ggg a$.*

*The behavioral profile of the net is the set of relations* $\{\rightsquigarrow, \nleftarrow, +, ||\}$.

**Definition 6** (Behavioral Profile-based Rules ($\beta_{\mathcal{N}}$))**.** *Let $\mathcal{N}$ = $(\mathcal{P}, \mathcal{T}, \mathcal{F}, \lambda)$ be a Petri net with $\{\rightsquigarrow, \nleftarrow, +, ||\}$ the behavioral profile. Let $a \in \mathcal{U}_{\mathcal{A}}$. We define behavioral-profile based rules $\beta_a : {\mathcal{U}_{\mathcal{A}}}^* \rightarrow \{\top, \bot\}$ such that for $\sigma \in {\mathcal{U}_{\mathcal{A}}}^*$ holds:*

$\beta_a(\sigma) = \neg\exists_{1 \leq j < i \leq |\sigma|} \sigma[i] = a \wedge (\sigma[j] \nleftarrow a \vee a + \sigma[j])$

*Like before, we use $\beta_{\mathcal{N}}$ to denote the full set of rules, i.e. $\beta_{\mathcal{N}} = \{\beta_a \mid a \in \mathcal{U}_{\mathcal{A}}\}$.*

A splitpoint is defined at an index $i$ in the trace if there is a rule that is violated at index $i$ (different from before, the same rule may trigger multiple consecutive violations). As events are checked according to their order in a trace, an error arises when the relation between any previous activity and the last activity in the trace is *reverse strict order* (the new activity should occur before) or *exclusive* (both activities should not happen together). The former violation indicates there are swapped events, while the latter indicates there is an additional event. Note that behavioral profile-based rules cannot detect a missing event since the (reverse) strict relation only checks the order, but does not enforce the occurrence of another activity.

**Definition 7** (Splitpoints based on $\beta_{\mathcal{N}}$ ($sb_\sigma$))**.** *Let $\mathcal{L}^E \in \mathcal{B}({\mathcal{U}_{\mathcal{A}}}^*)$ be an event log and $\sigma$ is a trace in $\mathcal{L}^E$. Let $\mathcal{N}$ = $(\mathcal{P}, \mathcal{T}, \mathcal{F}, \lambda)$ be a Petri net.*

*The set of splitpoints* $\text{sb}_\sigma$ *for $\sigma$ is defined as follows. For $1 \leq i \leq |\sigma|$ holds that $i \in \text{sb}_\sigma$ if and only if there exists an $a \in \mathcal{U}_{\mathcal{A}}$ with $\beta_a(\sigma[1..i]) = \bot$.*

Consider again a Petri net $\mathcal{N}$ shown in Figure 1. Let $\sigma = \langle a, d, b, c, b, e\rangle$ be a trace to be aligned with $\mathcal{N}$. From $\beta_{\mathcal{N}}$ we know that if $b$ and $d$ exist, then $b$ must appear before $d$, i.e. $d \nleftarrow b$. Similarly, $c$ and $d$ are in reverse strict order relation, i.e. $d \nleftarrow c$. However, the 3rd, 4th, and 5th events in $\sigma$ do not comply with these rules as shown in Table II. Therefore, the splitpoints in $\sigma$ are $sb_\sigma = \{3, 4, 5\}$. (Note that for the example in Table I, the set $sb_\sigma = \emptyset$, i.e. no behavioral rules are violated in that trace.)

TABLE II
CHECKING A TRACE $\sigma = \langle a, d, b, c, b, e\rangle$ BASED ON BEHVIOURAL-PROFILE-BASED RULES LEADS TO THE SPLITPOINTS $sb_\sigma = \{3, 4, 5\}$ (NOTE $sp_\sigma = \{2\}$ FOR THIS TRACE).

| $\beta_{\mathcal{N}}$ | $\sigma =$ $\langle a$ | $d$ | $b$ | $c$ | $b$ | $e\rangle$ |
|---|---|---|---|---|---|---|
| $d \nleftarrow b$ | | | $\bot$ $_{d \not\gg b}$ | | $\bot$ $_{d \not\gg b}$ | |
| $d \nleftarrow c$ | | | | $\bot$ $_{d \not\gg c}$ | | |
| splitpoints: | | | 3 | 4 | 5 | |

## C. Differences between rule sets

We have presented two different techniques to determine splitpoints. In Table III we summarize how the two techniques locate splitpoints given three different types of error. Here we use the same Petri net as displayed in Figure 1 and color the splitpoints with red.

The first row of Table III shows splitpoints for a trace with a missing event. A missing event is detected by the rules in $\rho_{\mathcal{N}}$ but not by the rules in $\beta_{\mathcal{N}}$. Due to the way alignments are computed, not flagging missing events is a positive point. The alignment algorithm is generally able to detect missing events easily and hence introducing splitpoints for missing events unnecessarily complicates the alignment algorithm.

The second row of Table III shows splitpoints in swapped events. Two swapped events ($e$ and $d$) are treated differently. $\rho_{\mathcal{N}}$ marks the first event ($e$), while $\beta_{\mathcal{N}}$ marks the second event ($d$). Finally, the third row of Table III shows splitpoints in additional events. An additional event is flagged by $\rho_{\mathcal{N}}$ but not by $\beta_{\mathcal{N}}$. However, in $\beta_{\mathcal{N}}$, the additional event propagates other errors ($b$ and $d$). This is due to the fact that $\beta_{\mathcal{N}}$ only checks order but not occurrence, hence an additional event with correct order is not detected as an error. Consequently, the correct next events are mistakenly detected as swapped events.

In conclusion, both techniques identify different sets of events and in Section VI we compare the performance of the alignment technique using any of the two sets as input, as well as the performance when combining both sets.

TABLE III
DIFFERENCE BETWEEN $\rho_{\mathcal{N}}$ AND $\beta_{\mathcal{N}}$ IN HANDLING ERROR.

| Type of Error | Error | $\rho_{\mathcal{N}}$ | $\beta_{\mathcal{N}}$ |
|---|---|---|---|
| Missing event | Missing $d$ before $e$ | $\langle a, b, e\rangle$ | $\langle a, b, e\rangle$ |
| Swapped event | $d$ and $e$ are swapped | $\langle a, b, e, d\rangle$ | $\langle a, b, e, d\rangle$ |
| Additional event | Additional $d$ | $\langle a, d, b, d, e\rangle$ | $\langle a, d, b, d, e\rangle$ |

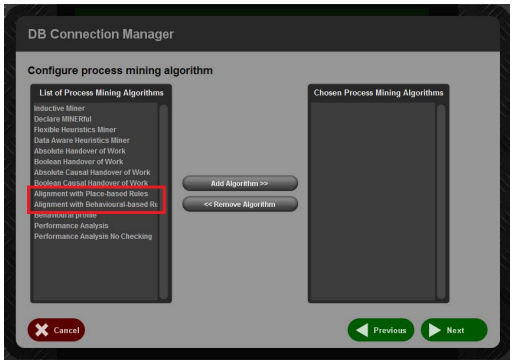



Fig. 5. Two alignment approaches $\rho_{\mathcal{N}}$ and $\beta_{\mathcal{N}}$ can be chosen in ProM.

## V. Implementation

The work in this paper is embedded in a larger project where the focus is on in-database preprocessing of event data for the purpose of speeding up the computation time needed while doing process mining [15], [16]. Therefore, we implemented the work into a dedicated H2 database system which is preloaded with the two rule sets for each process model in the evaluation. To achieve this, we modified the H2 database engine[2] and built a native operator [14] for rule checking. The native operator performs an automatic checking mechanism for each inserted event to determine splitpoints. On the ProM side, we have implemented this work as two ProM plug-ins: *IncPM Database Manager* and *Mine with IncPM*[3].

In general, there are two steps in alignments with predefined splitpoints. The first step is preprocessing, which includes establishing a database connection and producing rules based on Petri net places or behavioral profiles. We use the first plug-in for this step to produce a controller object containing database connection details and the rules. The next step is the main process executed by the second plug-in. Once we connect the controller object with the plug-in, the H2 database starts the rule checking process and produces splitpoints. When alignments are started, the log and splitpoints tables are imported to process mining tool to accelerate the computation. We exploit the alignment algorithm introduced in [19] for aligning the log and the model.

## VI. Evaluation

### A. Alignment Performance

We conducted several experiments that aim to investigate alignments performance in terms of computation time and memory usage between: (1) alignments with place-based rules ($\rho_{\mathcal{N}}$), (2) alignments with behavioral profile-based rules ($\beta_{\mathcal{N}}$), (3) $\rho_{\mathcal{N}}$ and $\beta_{\mathcal{N}}$ ($\rho_{\mathcal{N}} \wedge \beta_{\mathcal{N}}$), (4) $\rho_{\mathcal{N}}$ or $\beta_{\mathcal{N}}$ ($\rho_{\mathcal{N}} \vee \beta_{\mathcal{N}}$), (5) incremental alignments (*Inc*), and (6) alignments with standard A$^\star$ algorithm ($A^\star$). We applied those techniques to seven artificial large logs, four real-life logs [5], [9], [10], [18], and seven benchmark logs [11] [4]. The artificial logs are divided into two sets: the first set is designed to contain 20% swapped events, while the second set is designed to contain 20% missing events and 20% additional events. We used ProM plug-ins entitled "Swap Events in Log Traces" and "Add Noise to Log Filter" to build these logs.

We measured the computation time (in seconds), memory usage (in KB), and the number of alignments that timeout, i.e. that cannot be computed within 30 clock seconds. We also measured the preprocessing time that is needed to compute the sets of rules and to evaluate these rules on the event log. Furthermore, we recorded the information about the log and model characteristics, the average number of splitpoints per trace, and the total number of generated rules. The results are displayed in Table IV and Table V. Within those tables, the lowest number is highlighted with light gray color, while the highest is highlighted with dark gray.

The dash symbol in the table represents the fact that there is no value provided. This is the case for a number of large artificial models. The reason for this is that computing behavioral profiles for these logs is a non-trivial task. Here, we rely on the work presented in [3] which uses unfoldings to compute a finite a-cyclic representation of a Petri net on which behavioral profiles can be computed. Unfortunately, for these models, this work is unable to provide such unfoldings in reasonable time due to the combination of loops and many choices in these models. The same applies to the Sepsis and BPI2012 process models. However, as these models are simpler, the behavioral profiles were made manually.

The number of generated rules in Table V shows that $\beta_{\mathcal{N}}$ rules are more than $\rho_{\mathcal{N}}$ rules as each combination of activity pairs contributes one rule in $\beta_{\mathcal{N}}$. Moreover, the large number in $\rho_{\mathcal{N}}$ rules of the Sepsis log indicates that there are some overlapping rules. We aim to fix this in the future work.

From Table IV we can see that the preprocessing time to predefine splitpoints is insignificant compared to the computation time of alignments. In fact, the preprocessing step can be done offline (i.e. automatic in the database), hence it does not add to the actual time of alignments. Furthermore, we can see that alignments with predefined splitpoints ($\rho_{\mathcal{N}}$, $\beta_{\mathcal{N}}$, $\rho_{\mathcal{N}} \wedge \beta_{\mathcal{N}}$, and $\rho_{\mathcal{N}} \vee \beta_{\mathcal{N}}$) outperform *Inc* and $A^\star$ in most of the logs. There are only two logs where $A^\star$ shows the fastest computation time. The first log is the artificial log prEm6. This log contains traces where the initial and final labels are swapped. This type of error is easily detected by alignments without extending the marking equation as the A$^\star$ algorithm will detect the swap after executing the first event in the log and is then able to produce the alignment for the fitting model with just one additional heuristic computation.

The second log where $A^\star$ is relatively fast is Sepsis. In this log, there are only 1050 traces with 16 activities and hardly any parallelism in the model. With such small numbers of activities, the overhead of using more complex heuristic functions (which is the result of introducing splitpoints) is higher than the time

[2]https://github.com/alifahsyamsiyah/h2processmining-master

[3]https://svn.win.tue.nl/repos/prom/Packages/InDatabasePreprocessing/

[4]See https://github.com/alifahsyamsiyah/ICPM2019Experiment/

TABLE IV

LOG AND MODEL CHARACTERISTICS, PREPROCESSING TIME, AND COMPUTATION TIME (IN SECONDS) OF ALIGNMENTS.

| Log | #Trace | #Act | #Place | #Trans | Preprocessing time | | Alignments computation time | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | $\rho_{\mathcal{N}}$ | $\beta_{\mathcal{N}}$ | $\rho_{\mathcal{N}}$ | $\beta_{\mathcal{N}}$ | $\rho_{\mathcal{N}} \wedge \beta_{\mathcal{N}}$ | $\rho_{\mathcal{N}} \vee \beta_{\mathcal{N}}$ | *Inc* | $A^\star$ |
| L1 - swap | 100 000 | 20 | 21 | 20 | 3.298 | 3.686 | 1157.648 | 1859.426 | 1775.274 | 1478.271 | 3626.408 | 24 633.656 |
| L2 - swap | 500 000 | 8 | 8 | 8 | 2.891 | 3.743 | 848.768 | 851.896 | 1093.964 | 815.720 | 2208.082 | 1240.848 |
| L3 - swap | 500 000 | 11 | 11 | 11 | 13.511 | 3.525 | 1152.997 | 1170.952 | 1396.992 | 1190.423 | 3283.316 | 2159.843 |
| L4 - swap | 500 000 | 8 | 6 | 8 | 2.766 | 8.585 | 1070.663 | 1084.864 | 1370.259 | 919.620 | 2678.945 | 1381.094 |
| L5 - miss, add | 100 000 | 8 | 8 | 8 | 0.719 | 2.327 | 154.533 | 160.091 | 177.030 | 148.596 | 248.154 | 167.829 |
| L6 - miss, add | 100 000 | 11 | 11 | 11 | 1.328 | 1.261 | 188.224 | 180.059 | 210.623 | 185.992 | 299.501 | 207.817 |
| L7 - miss, add | 100 000 | 8 | 6 | 8 | 0.657 | 4.302 | 186.673 | 208.455 | 231.307 | 180.160 | 316.901 | 194.938 |
| L8 - Roadfine | 150 370 | 11 | 17 | 15 | 3.371 | 1.431 | 160.586 | 152.305 | 155.510 | 159.630 | 200.709 | 200.299 |
| L9 - BPI2012 | 13 087 | 36 | 22 | 37 | 0.480 | 0.219* | 389.586 | 461.983 | 459.933 | 387.377 | 471.748 | 585.613 |
| L10 - Sepsis | 1050 | 16 | 25 | 26 | 0.516 | 0.016* | 24.804 | 58.430 | 63.310 | 24.949 | 61.565 | 6.312 |
| L11 - Hospital | 100 000 | 18 | 12 | 14 | 1.180 | 1.221 | 105.275 | 103.430 | 103.334 | 104.148 | 135.153 | 130.778 |
| L12 - prAm6 | 1200 | 726 | 347 | 363 | 2.961 | - | 65.678 | - | - | - | 297.183 | 322.937 |
| L13 - prBm6 | 1200 | 317 | 317 | 317 | 1.765 | - | 6.919 | - | - | - | 7.512 | 7.168 |
| L14 - prCm6 | 500 | 311 | 317 | 317 | 1.109 | - | 179.247 | - | - | - | 744.819 | 1887.621 |
| L15 - prDm6 | 1200 | 429 | 529 | 429 | 18.064 | - | 2499.406 | - | - | - | 4932.314 | 14 203.424 |
| L16 - prEm6 | 1200 | 275 | 277 | 275 | 3.236 | - | 44.163 | - | - | - | 44.112 | 42.784 |
| L17 - prFm6 | 1200 | 299 | 362 | 299 | 11.117 | - | 694.288 | - | - | - | 1764.526 | 15 472.373 |
| L18 - prGm6 | 1200 | 335 | 357 | 335 | 6.693 | - | 385.704 | - | - | - | 1470.476 | 35 982.940 |

TABLE V

TOTAL NUMBER OF GENERATED RULES, MEMORY USAGE (IN KB), TIMEOUTS, AND AVERAGE NUMBER OF SPLITPOINTS PER TRACE IN ALIGNMENTS

| Log | $\rho_{\mathcal{N}}$ | $\beta_{\mathcal{N}}$ | $\rho_{\mathcal{N}}$ | | | $\beta_{\mathcal{N}}$ | | | $\rho_{\mathcal{N}} \wedge \beta_{\mathcal{N}}$ | | | $\rho_{\mathcal{N}} \vee \beta_{\mathcal{N}}$ | | | *Inc* | | | $A^\star$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | #Rules | #Rules | Mem | TO | Avg spl. | Mem | TO | Avg spl. | Mem | TO | Avg spl. | Mem | TO | Avg spl. | Mem | TO | Avg spl. | Mem | TO |
| L1 | 20 | 116 | 827 | 0 | 7.703 | 2700 | 0 | 10.713 | 1023 | 0 | 3.641 | 3001 | 0 | 14.775 | 1146 | 0 | 8.011 | 254 | 0 |
| L2 | 7 | 29 | 186 | 0 | 2.953 | 337 | 0 | 3.495 | 166 | 0 | 1.860 | 363 | 0 | 4.588 | 143 | 2 | 2.883 | 18 | 1 |
| L3 | 10 | 64 | 203 | 0 | 3.740 | 356 | 0 | 5.320 | 178 | 0 | 2.379 | 456 | 0 | 6.680 | 195 | 0 | 3.817 | 19 | 0 |
| L4 | 5 | 27 | 153 | 0 | 3.124 | 390 | 0 | 3.751 | 146 | 0 | 1.975 | 402 | 0 | 4.900 | 166 | 2 | 3.337 | 15 | 1 |
| L5 | 7 | 29 | 102 | 0 | 2.769 | 214 | 0 | 1.601 | 98 | 0 | 1.277 | 235 | 0 | 3.094 | 76 | 0 | 1.685 | 14 | 0 |
| L6 | 10 | 64 | 163 | 0 | 3.478 | 304 | 0 | 2.087 | 150 | 0 | 1.473 | 318 | 0 | 4.092 | 136 | 0 | 1.899 | 17 | 0 |
| L7 | 5 | 27 | 210 | 0 | 2.993 | 196 | 0 | 1.654 | 154 | 0 | 1.302 | 221 | 0 | 3.346 | 151 | 0 | 2.066 | 16 | 0 |
| L8 | 14 | 57 | 21 | 0 | 1.585 | 29 | 0 | 1.079 | 16 | 0 | 1.001 | 38 | 0 | 1.663 | 16 | 0 | 1.029 | 9 | 0 |
| L9 | 14 | 47 | 1018 | 0 | 2.811 | 960 | 0 | 1 | 960 | 0 | 1 | 1018 | 0 | 2.811 | 1049 | 0 | 4.954 | 137 | 0 |
| L10 | 104 | 88 | 645 | 0 | 2.693 | 931 | 1 | 1.355 | 990 | 1 | 1.001 | 550 | 0 | 3.048 | 1049 | 0 | 1.736 | 176 | 0 |
| L11 | 11 | 21 | 111 | 0 | 1.047 | 75 | 0 | 1.075 | 86 | 0 | 1.011 | 111 | 0 | 1.112 | 75 | 0 | 1.025 | 53 | 0 |
| L12 | 346 | - | 1348 | 0 | 4.029 | - | - | - | - | - | - | - | - | - | 2210 | 0 | 3.604 | 70 | 0 |
| L13 | 316 | - | 81 | 0 | 1.022 | - | - | - | - | - | - | - | - | - | 53 | 0 | 1 | 57 | 0 |
| L14 | 316 | - | 5062 | 0 | 12.872 | - | - | - | - | - | - | - | - | - | 5072 | 0 | 13.250 | 332 | 0 |
| L15 | 528 | - | 9713 | 0 | 11.753 | - | - | - | - | - | - | - | - | - | 13 886 | 2 | 6.674 | 1931 | 468 |
| L16 | 276 | - | 275 | 0 | 3.522 | - | - | - | - | - | - | - | - | - | 159 | 0 | 2 | 122 | 0 |
| L17 | 361 | - | 3386 | 0 | 12.657 | - | - | - | - | - | - | - | - | - | 2675 | 0 | 6.507 | 1648 | 514 |
| L18 | 356 | - | 2161 | 0 | 11.576 | - | - | - | - | - | - | - | - | - | 1882 | 0 | 10.342 | 3358 | 1200 |

needed to simply visit the entire search space, and therefore $A^\star$ is fastest.

However, for the logs with more intricate errors and lots of activity labels, $A^\star$ is unable to perform better than the others, and in many cases it may even not finish the alignment computation within the boundary time. We can see there are quite large numbers of timeouts in $A^\star$, e.g. there are 1200 timeouts in prGm6 which means all traces in the log cannot be aligned within the time limit.

We also observed that *Inc* is not better than $A^\star$ in some of the logs, specifically in logs with small numbers of activities. This indicates that splitting in the middle of the search may not help the alignments to find the solution faster, i.e. the overhead of finding each splitpoint incrementally is higher than the time it takes to simply search through the entire search space. Unlike alignments with predefined splitpoints, *Inc* has to investigate large portions of the search space before concluding that there should be additional splitpoints in the remainder of the trace. As a result, *Inc* produces some timeouts in prDm6 and some other logs with long traces.

Among four techniques with predefined splitpoints, $\rho_{\mathcal{N}} \wedge \beta_{\mathcal{N}}$ shows the longest computation time in most of the logs. $\rho_{\mathcal{N}} \wedge \beta_{\mathcal{N}}$ takes the intersection of splitpoints in $\rho_{\mathcal{N}}$ and $\beta_{\mathcal{N}}$, i.e. it only concludes that there is a splitpoint if both sets of rules agree on that. Hence, this makes the number of splitpoints in $\rho_{\mathcal{N}} \wedge \beta_{\mathcal{N}}$ lower compared to $\rho_{\mathcal{N}}$, $\beta_{\mathcal{N}}$, and $\rho_{\mathcal{N}} \vee \beta_{\mathcal{N}}$. In the paper we already saw some examples where both sets of rules completely disagree, thus leading to the same time complexity as *Inc*.

In contrast, $\rho_{\mathcal{N}} \vee \beta_{\mathcal{N}}$ takes the union of splitpoints in $\rho_{\mathcal{N}}$ and $\beta_{\mathcal{N}}$, i.e. it concludes that there is a splitpoint if any of the rules indicate this. Although this design choice boosts the alignments performance, it comes at a cost of memory (see Table V). However, as all numbers in Table V indicate the maximum amount of memory in kB needed to align a single trace, there is little cause for concern. Traditionally, $A^\star$ requires the least amount of memory as most memory is needed in the computation of the heuristic function values. Only in case all traces timeout, the size of the search space becomes larger than the memory needed to compute the heuristic, thus leading to the fact that for prGm6, *Inc* uses the least amount of memory.

Next to the two combinations of $\rho_{\mathcal{N}}$ and $\beta_{\mathcal{N}}$, we also included both separately in the experiments. There are no obvious consistent differences in computation time for either approach. However, $\beta_{\mathcal{N}}$ uses more memory, indicating it identifies more splitpoint upfront. Nonetheless, $\rho_{\mathcal{N}}$ can be 4 times faster than *Inc*, which is currently considered state-of-the-art, and the best performance gain is in the log prGm6 where $\rho_{\mathcal{N}}$ is 93 times faster than $A^\star$.

In conclusion, predefined splitpoints indeed help alignments to perform faster compared to *Inc* and $A^\star$. Based on these experiments, we cannot conclude that any of the two sets of rules leads to better results in general. However, as it is not always possible to compute the behavioral profiles upfront, we recommend sticking to $\rho_{\mathcal{N}}$ as the method of choice for computing alignments.

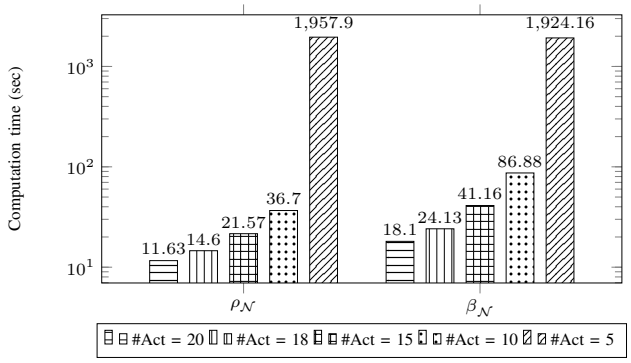



Fig. 6. Comparison of $\rho_{\mathcal{N}}$and $\beta_{\mathcal{N}}$in Petri net with duplicate labels.

### B. Alignments with Duplicate Labels

In this experiment, we aim to investigate the effect of duplicate labels. To this end, we took the first model of Table IV which contains 20 transitions associated to 20 activities, i.e. there are no duplicate labels. Using this model as a basis, we created four new models by relabelling transitions such that the number of activities decreases to 18, 15, 10 and 5. Furthermore, if a transition label is changed, the corresponding activity label in the log is also changed. We plot the result in Figure 6.

As expected, this experiment shows that duplicate labels deteriorate the rule quality of $\rho_{\mathcal{N}}$ and $\beta_{\mathcal{N}}$. As mentioned earlier, $\rho_{\mathcal{N}}$ treats duplicate labels by aggregating rules pertaining to them. Consequently, the more duplicate labels, the weaker rules resulted, thus the less errors are detected. In addition, multiple occurrences of one activity label cause this label to be more likely to be interleaved with other labels. As a result, the rules in $\beta_{\mathcal{N}}$, which only identify violations against exclusiveness and reverse strict order relations, also become less restrictive. In the last model we can see the computation time between $\rho_{\mathcal{N}}$ and $\beta_{\mathcal{N}}$ become similar as there is only few splitpoint differences in both techniques. In the worst case, however, the initially identified splitpoints are empty and the techniques become identical to *Inc*.

## VII. Conclusion

This paper proposes an approach to improve alignment computation through predefined splitpoints. Splitpoints are a set of event indexes in which the events fail to pass a rule checking mechanism. During checking, we exploit rules which represent a process model in terms of structural or behavioral properties. The former utilizes places of a model while the latter utilizes behavioral profiles of a model.

The rule checking is performed in a data source where events are stored and it is done automatically and immediately upon event insertion. Therefore, the splitpoints are available before starting an alignment computation. When alignment computation begins, all three necessary elements are imported: a process model, a log, and a set of splitpoints. To perform an alignment, we specifically utilize the work in [20] which extends the marking equation to make use of the splitpoints.

We evaluated this approach on real-life logs as well as benchmarking logs. The results show that predefined splitpoints indeed help alignments to perform faster compared to the incremental alignments [20] and the classical alignments with tuned parameters [21]. Moreover, both place-based and behavioral profile-based rules provide improvement.

For future work we aim to reduce overlapping rules and fix the known limitation in invisible transitions with self loop.

## References


[1] A. Adriansyah. *Aligning Observed and Modeled Behavior*. PhD thesis, TU Eindhoven, 2014.

[2] Vincent Bloemen, S.J. van Zelst, W.M.P. van der Aalst, B.F. van Dongen, and Jaco van de Pol. Maximizing synchronization for aligning observed and modelled behaviour. In *Proceedings of the BPM Conference 2018, Sydney, NSW, Australia*, pages 233–249, 2018.

[3] A. Burattin, S.J. van Zelst, A. Armas-Cervantes, B.F. van Dongen, and J. Carmona. Online Conformance Checking Using Behavioural Patterns. *Business Process Management*, pages 250–267, 2018.

[4] J. Carmona, B.F. van Dongen, A. Solti, and M. Weidlich. *Conformance Checking - Relating Processes and Models*. Springer, 2018.

[5] M. De Leoni and F. Mannhardt. Road Traffic Fine Management Process, DOI:10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5 , 2015.

[6] M. de Leoni and A. Marrella. Aligning real process executions and prescriptive process models through automated planning. *Expert Systems with Applications*, 82:162 – 183, 2017.

[7] W.L. Jonathan Lee, H.M.W. Verbeek, J. Munoz-Gama, W.M.P. van der Aalst, and M. Sepúlveda. Recomposing Conformance: Closing the Circle on Decomposed Alignment-based Conformance Checking in Process Mining. *Inf. Sci.*, 466:55–91, 2018.

[8] María Teresa Gómez López, Diana Borrego, Josep Carmona, and Rafael M. Gasca. Computing alignments with constraint programming: The acyclic case. In *Proceedings of ATAED 2016, Torun, Poland, June 20-21, 2016.*, pages 96–110, 2016.

[9] F. Mannhardt. Sepsis Cases - Event Log, DOI: 10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460, 2016.

[10] F. Mannhardt. Hospital Billing - Event Log, DOI: 10.4121/uuid:76c46b83-c930-4798-a1c9-4be94dfeb741, 2017.

[11] J. Munoz-Gama. Conformance Checking in the Large (BPM 2013), DOI: 10.4121/UUID:44C32783-15D0-4DBD-AF8A-78B97BE3DE49, 2013.

[12] J. Munoz-Gama, J. Carmona, and W.M.P. van Der Aalst. Single-Entry Single-Exit Decomposed Conformance Checking. *Inf. Syst.*, 46:102–122, December 2014.

[13] Daniel Reißner, Raffaele Conforti, Marlon Dumas, Marcello La Rosa, and Abel Armas-Cervantes. Scalable conformance checking of business processes. In *OTM 2017 Conferences*, 2017.

[14] A. Syamsiyah, B.F. van Dongen, and R. Dijkman. A Native Operator for Process Discovery. *DEXA 2018*, pages 292–300, 2018.

[15] A. Syamsiyah, B.F. van Dongen, and W.M.P. van der Aalst. DB-XES: Enabling Process Mining in the Large. In *SIMPDA 2016 - Extended Versions*, pages 63–77, 2016.

[16] A. Syamsiyah, B.F. van Dongen, and W.M.P van der Aalst. Recurrent Process Mining with Live Event Data. In *BPI 2017*, pages 178–190, 2017.

[17] F. Taymouri and J. Carmona. A Recursive Paradigm for Aligning Observed Behavior of Large Structured Process Models. *BPM 2016, Rio de Janeiro, Brazil, Proceedings*, pages 197–214, 2016.

[18] B.F. van Dongen. BPI Challenge 2012, DOI: 10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f, 2012.

[19] B.F. van Dongen. Efficiently Computing Alignments - Algorithm and Datastructures. *BPM Workshops 2018, Sydney, Australia, Revised Papers*, pages 44–55, 2018.

[20] B.F. van Dongen. Efficiently Computing Alignments - Using the Extended Marking Equation. *BPM 2018, Sydney, Australia, Proceedings*, pages 197–214, 2018.

[21] S.J. van Zelst, A. Bolt, and B.F. van Dongen. Tuning Alignment Computation: An Experimental Evaluation. *Proceedings of ATAED 2017, Zaragoza, Spain, June 26-27, 2017.*, pages 6–20, 2017.

[22] M. Weidlich, A. Polyvyanyy, N. Desai, J. Mendling, and M. Weske. Process Compliance Analysis Based on Behavioural Profiles. *Inf. Syst.*, 36(7):1009–1025, November 2011.

# Monotone Conformance Checking for Partially Matching Designed and Observed Processes

Artem Polyvyanyy
School of Computing and Information Systems
The University of Melbourne
Email: artem.polyvyanyy@unimelb.edu.au

Anna Kalenkova
Laboratory of Process-Aware Information Systems
Higher School of Economics
Email: akalenkova@hse.ru

***Abstract*—Conformance checking is a subarea of process mining that studies relations between designed processes, also called process models, and records of observed processes, also called event logs. In the last decade, research in conformance checking has proposed a plethora of techniques for characterizing the discrepancies between process models and event logs. Often, these techniques are also applied to measure the quality of process models automatically discovered from event logs. Recently, the process mining community has initiated a discussion on the desired properties of such measures. This discussion witnesses the lack of measures with the desired properties and the lack of properties intended for measures that support partially matching processes, i.e., processes that are not identical but differ in some steps. The paper at hand addresses these limitations. Firstly, it extends the recently introduced precision and recall conformance measures between process models and event logs that possess the desired property of monotonicity with the support of partially matching processes. Secondly, it introduces new intuitively desired properties of conformance measures that support partially matching processes and shows that our measures indeed possess them. The new measures have been implemented in a publicly available tool. The reported qualitative and quantitative evaluations based on our implementation demonstrate the feasibility of using the proposed measures in industrial settings.**

**Keywords: Process mining, conformance checking, entropy, partial matching, monotonicity.**

## I. Introduction

Process mining aims to discover, monitor, and improve processes *observed* in the real world using the knowledge accumulated in event logs produced by executions of *designed* processes implemented in modern information systems [1], where an event log is a collection of recorded *traces* each capturing an instance of an executed business process. Process discovery and conformance checking are two central subareas in process mining. Process discovery studies methods, techniques, and tools to automatically construct a "good" process model from an event log to aggregate and analyze the knowledge about business processes as they were observed in the real world [2]–[4]. Conformance checking studies methods, techniques, and tools to characterize and quantify differences between an event log and process model [5]–[8], e.g., to measure and explain the "goodness" of the model (constructed or discovered) that encodes the traces from the event log.

Two basic measures in conformance checking are *precision* and *recall* (a.k.a. *fitness*). Precision aims to characterize the relation between the amount of process information shared by the traces of an event log and process model and the amount of process information encoded in the traces of the model. Recall, in turn, strives to characterize the relation between the shared information and the information about processes encoded in the event log traces. The precision and recall are usually defined as numeric quantities that take values between (and including) zero and one. The precision and recall values of zero represent a situation of no similarities between the traces of the event log and the model traces. The greater the precision and recall values, the greater the similarity between the event log and the process model traces. Finally, the precision and recall values of one represent the situation when the event log and process model capture the same behavior.

Precision and recall are central performance measures for information retrieval systems [9]. Given a set of relevant documents and a set of documents retrieved/discovered by an information retrieval system, *precision* is the fraction of relevant retrieved documents over the retrieved documents, whereas *recall* is the fraction of relevant retrieved documents over the relevant documents. Similarly, given traces recorded in an event log (a.k.a. relevant traces) and a set of traces described by a process model discovered from the event log, e.g., using techniques proposed in [2]–[4], (a.k.a. discovered traces), *precision* is the fraction of relevant discovered traces over the discovered traces, whereas *recall* is the fraction of relevant discovered traces over the relevant traces.

Despite being conceptually clean and intuitive, the proposed above conformance measures of precision and recall face at least one major challenge. Unlike in information retrieval, where both collections of relevant and retrieved documents are finite, the set of traces encoded in a (discovered) process model is often infinite. This makes it practically difficult to define conformance measures with the desired properties, e.g., determinism and monotonicity, as it is not immediate to define how to measure infinite collections of traces. In [8], we overcame this challenge by proposing process conformance measures of precision and recall based on the notion of *topological entropy* over regular languages.

Although the entropy-based precision and recall are deterministic and monotonic measures, they perform unsatisfactorily in the presence of non-identical traces, even when traces differ only in a single process step. For instance, if each trace in an event log is *not* described in a process

978-1-7281-0919-0/19/$31.00 ©2019 IEEE
DOI 10.1109/ICPM.2019.00022

model but differs with some trace of the model in only one process step, precision and recall values between the event log and model are equal to zero, just like in the situation when these traces are completely different. In this paper, we overcome this limitation. We achieve this by "diluting" the traces captured in the compared event log and process model and then comparing these diluted traces. In our approach, the dilution of a trace results in the inclusion of all its sub-traces into the collection of traces captured in the corresponding event log or process model. We demonstrate that the extended measures possess intuitively desired properties for precision and recall in the presence of partially matching traces, viz. the more (in the number of traces) and the longer (in the number of matched process steps) are the partial matches between the traces captured in an event log and process model, the greater are the precision and recall values between them.

To summarize, this paper makes these contributions:

- Extends the entropy-based precision and recall measures proposed in [8] to support the partial matching between traces via comparisons of their sub-traces;
- Introduces, for the first time, properties for conformance measures that address the partial matching of traces; note that all the so far introduced properties for conformance measures [8], [10], [11] only address the comparisons of identical traces; and
- Reports on qualitative and quantitative evaluations of the proposed precision and recall measures that demonstrate the feasibility of using them in industrial settings.

The next section gives a motivating example. Section III introduces the basic notions. Section IV presents the entropy-based measures, whereas Section V extends them with partial matching support. Section VI presents the results of our evaluation, before stating concluding remarks.

## II. Motivating Example

Consider a simple process of booking a flight ticket captured as a finite automaton in Figure 1a. The process starts with the user opening a booking window. Then, she fills her name and passport data (in any order). Finally, the booking is confirmed and the window is closed. Suppose that the user behavior in the real-world can deviate from the one specified in the reference model. For example, a user may insert the name twice and close the window without the confirmation; this sequence of steps is captured in the automaton shown in Figure 1b. Despite the fact that the automata presented in Figure 1 describe similar traces, precision and recall measures that rely on exact comparisons of traces cannot appraise this similarity (i.e., the corresponding values of precision and recall equal to zero).

To address this limitation, we observe that the traces captured in the example automata coincide if certain steps get replaced with silent ($\tau$) steps. If one "mirrors" step $confirm$ in the automaton in Figure 1a and one step $fill\ name$ (any of the two) in the only trace of the automaton in Figure 1b with silent steps, cf. the dashed transitions in the figure, then both resulting automata describe the observable trace

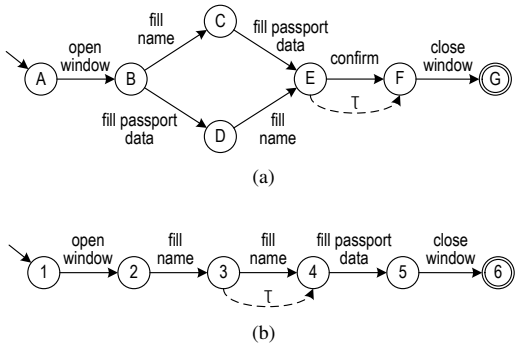



Figure 1: Two finite automata.

$\langle open\ window, fill\ name, fill\ passport\ data, close\ window \rangle$. This idea for identifying partially matching traces through their "dilution" with silent steps lies at the heart of the solution proposed in the work at hand.

According to the conformance measures proposed in this paper, precision and recall values between the automata in Figure 1 are 0.859 and 0.694, respectively. If one inserts step $confirm$ between $fill\ passport\ data$ and $close\ window$ in the only trace of the automaton in Figure 1b, i.e., makes it more similar to the traces of the automaton in Figure 1a, the values of precision and recall increase to 0.978 and 0.973, respectively, capturing the monotonic nature of our measures.

## III. Preliminaries

This section introduces basic notations and definitions.

### A. Sequences, Languages, and Event Logs

Let $X$ be a set of elements. A *power set* over set $X$, denoted as $\mathcal{P}(X)$, is the set of all subsets of $X$.

By $\langle x_1, x_2, \ldots, x_k \rangle$, where $x_1, x_2, \ldots, x_k \in X$, $k \in \mathbb{N}_0$, we denote a *sequence* of elements from $X$ of length $k$. The *empty sequence* of zero length is represented by $\langle\rangle$. A *concatenation* of two sequences $\langle x_1, x_2, \ldots, x_k \rangle$ and $\langle y_1, y_2, \ldots, y_l \rangle$ is denoted by $\langle x_1, x_2, \ldots, x_k \rangle \cdot \langle y_1, y_2, \ldots, y_l \rangle$ and equals to $\langle x_1, x_2, \ldots, x_k, y_1, y_2, \ldots, y_l \rangle$. $X^*$ designates the set of all finite sequences over $X$ (including the *empty sequence*).

An *alphabet* is any nonempty finite set. The elements of an alphabet are its *labels*. A (formal) *language* $L$ over an alphabet $\Sigma$ is a (not necessarily finite) set of *sequences*, or *words*, of elements from $\Sigma$, i.e., $L \subseteq \Sigma^*$. By $C_n(L)$, we denote the set of all words in $L$ of length $n$. By $\Xi$, we denote a universe of all possible *observable* labels. $\tau$ designates a special *silent* label, such that $\tau \notin \Xi$ and $\langle \tau \rangle = \langle\rangle$. Let $L_1$ and $L_2$ be two languages. Then, their concatenation is the language $\{l_1 \cdot l_2 \,|\, l_1 \in L_1, l_2 \in L_2\}$ denoted by $L_1 \circ L_2$.

Let $E$ be a finite nonempty set of events. A finite language $L \subseteq E^*$ is an *event log* and its words are called *traces*.

### B. Finite Automata

A *deterministic finite automaton*, or a DFA, is a 5-tuple $(Q, \Lambda, \delta, q_0, A)$, where $Q$ is a finite set of *states*, $\Lambda \subset \Xi$ is the *input alphabet*, such that $Q$ and $\Lambda$ are disjoint, $\delta : Q \times (\Lambda \cup$

$\{\tau\}) \to Q$ is the *transition function*, $q_0 \in Q$ is a *start state*, and $A \subseteq Q$ is a set of *accepting states*.

A *computation* of a DFA $B = (Q, \Lambda, \delta, q_0, A)$ is either the empty sequence or a sequence $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$, $n \in \mathbb{N}$, where $a_i \in \Lambda \cup \{\tau\}$, $i \in [1..n]$, and there is a sequence of states $\langle q_0, q_1, \ldots, q_n \rangle$, where $q_j \in Q$, $j \in [0..n]$, such that for every $k \in [1..n]$ it holds that $\delta(q_{k-1}, a_k) = q_k$. We say that $\sigma$ *leads to* $q_n$. By convention, the empty sequence always leads to the start state. Note that when referring to computations silent labels can be omitted, as $\langle a_1, a_2, ..., a_i, \tau, a_{i+1}, ..., a_k \rangle = \langle a_1, a_2, ..., a_i \rangle \cdot \langle \tau \rangle \cdot \langle a_{i+1}, ..., a_k \rangle = \langle a_1, a_2, ..., a_i \rangle \cdot \langle \rangle \cdot \langle a_{i+1}, ..., a_k \rangle = \langle a_1, a_2, ..., a_k \rangle$. In what follows, we only consider computations without silent labels. We say that $B$ *accepts* $\gamma \in \Lambda^*$ iff $\gamma$ is a computation of $B$ that leads to an accepting state $q \in A$. The *language* of $B$ is denoted by $lang(B)$ and is the set of all sequences $B$ accepts; we also say that $B$ *recognizes* $lang(B)$.

A DFA $(Q, \Lambda, \delta, q_0, A)$ is *ergodic* if its underlying graph is strongly irreducible, i.e., for all $(x, y) \in Q \times Q$ there exists a sequence of states $\langle q_1, \ldots, q_n \rangle \in Q^*$, $n \in \mathbb{N}$, such that $q_1 = x$, $q_n = y$, and for every $k \in [1 \,..\, n-1]$ there exists $\lambda \in \Lambda \cup \{\tau\}$ such that $\delta(q_k, \lambda) = q_{k+1}$.

A language $L \subseteq \Sigma^*$ is *regular* iff it is recognized by a DFA. $L \subseteq \Sigma^*$ is *irreducible* if, given two words $w_1$ and $w_2$ in $L$, there exists a word $w \in \Sigma^*$, such that $w_1 \cdot w \cdot w_2 \in L$. A regular language $L$ is irreducible iff it is a language of an ergodic DFA [12].

## IV. Entropy-based Conformance Checking

In this paper, we consider models such that their behaviors can be described in terms of DFAs. These can be Petri nets or BPMN models that induce finite reachability graphs, or (not necessary deterministic) finite automata, which can always be converted into equivalent DFAs [13]. As an event log induces a finite language, it can, as well, be encoded as a DFA.

To compute precision and recall between a model and event log, we compare the languages the corresponding DFAs recognize. Specifically, we measure the ration of the traces the model and log share to the traces captured in the model or log. To obtain such a measure over collections of traces, we estimate the cardinality of the corresponding languages, i.e., the number of words that belong to the languages.

Next, in Section IV-A, we present the notion of topological entropy. We use topological entropy to define the notion of the short-circuit entropy of a regular language (see Section IV-B), which, in turn, in Section IV-C, is used to define the entropy-based precision and recall between a model and log.

### A. Topological Entropy

The cardinality of a finite language $L \subseteq \Sigma^*$ can be naturally defined as $|L|$, i.e., the number of words in $L$. However, this definition is not particularly useful for infinite languages. One can use *topological entropy* to estimate the cardinality of an irreducible language $L$ [12], which is defined as follows:

$$ent(L) = \lim_{n \to \infty} \sup \frac{\log |C_n(L)|}{n}.$$

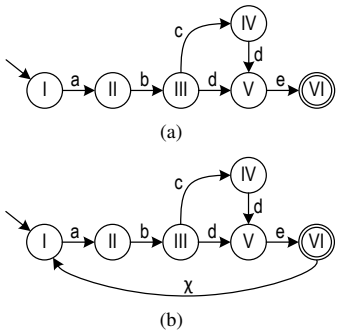



Figure 2: Two finite automata; (b) is ergodic.

Thus, the topological entropy of an irreducible language estimates the number of distinct words in the language with respect to the length of these words. One can compute the topological entropy of an irreducible language $L$ as the logarithm of the Perron-Frobenius eigenvalue of the adjacency matrix of a DFA that recognizes $L$ [12].

### B. Short-circuit Entropy

Because topological entropy is defined over irreducible languages, one cannot use it to estimate cardinality of infinite non-irreducible regular languages or finite languages, e.g., event logs. To allow estimating cardinality of an arbitrary regular language, in [8], we proposed the notion of a *short-circuit measure* over languages. Given a measure over languages $m : \mathcal{L} \to \mathbb{R}_0^+$, where $\mathcal{L} \subseteq \mathcal{P}(\Sigma^*)$ and $\Sigma \subset \Xi$, its *short-circuit* version $m\bullet$ is defined as $m\bullet(L) = m((L \circ \{\langle \chi \rangle\})^* \circ L)$, where $L \subseteq \Sigma^*$ and $\chi \in \Xi \setminus \Sigma$. Because given a regular language $L$, $(L \circ \{\langle \chi \rangle\})^* \circ L$ is irreducible [8], $ent\bullet(L)$ can always be computed.

Hence, to compute the *short-circuit (topological) entropy* of a regular language $L$, i.e., $ent\bullet(L)$, one can transform a DFA that recognizes $L$ by inserting additional transitions marked by a special label $\chi$ not observed in $L$ that connect all the accepting states of the DFA with its start state, and then compute the topological entropy of the language recognized by the resulting automaton. For example, $ent\bullet(L)$, where $L$ is the language recognized by the DFA in Figure 2a, can be computed as topological entropy of the language recognized by the DFA in Figure 2b.

It holds that $ent\bullet(\emptyset) = 0$. Moreover, for two regular languages $L_1$ and $L_2$, such that $L_1 \subset L_2$, it holds that $ent\bullet(L_1) < ent\bullet(L_2)$, refer to [8] for details. Next, we show two additional properties of the short-circuit entropy measure over regular languages. These properties of language measures are used to establish several new properties of the entropy-based precision and recall measures, which also apply for the new conformance measures that support partial matching of processes, refer to Section V.

If for every length, one regular language has no more words of that length than the other regular language, then the short-circuit entropy of the former language is less than or equal to the short-circuit entropy of the latter language.

**Theorem 1** (Monotonicity on the number of words)**.** *Let $L_1$ and $L_2$ be two regular languages such that $\forall\, k \in \mathbb{N} : |C_k(L_1)| \leq |C_k(L_2)|$. Then, $ent\bullet(L_1) \leq ent\bullet(L_2)$.*

*Proof.* Let $x_n = \frac{\log |C_n(L_1)|}{n}$ and $y_n = \frac{\log |C_n(L_2)|}{n}$, $n \geq 1$. Suppose that $\{x_{n_l}\}_{l=1}^{\infty}$, where $n_1 < n_2 < \ldots$, is a subsequence of $\{x_n\}_{n=1}^{\infty}$, such that $\lim_{l\to\infty} x_{n_l} = ent\bullet(L_1)$. Consider the corresponding sequence $\{y_{n_l}\}_{l=1}^{\infty}$. As follows from the theorem conditions, $\forall l \in \mathbb{N}, l \geq 1 : x_{n_l} \leq y_{n_l}$, then $\lim_{l\to\infty} \sup x_{n_l} \leq \lim_{l\to\infty} \sup y_{n_l}$. Hence, $ent\bullet(L_1) = \lim_{l\to\infty} x_{n_l} = \lim_{l\to\infty} \sup x_{n_l} \leq \lim_{l\to\infty} \sup y_{n_l} \leq ent\bullet(L_2)$. □

Next, we refine the above property.

**Theorem 2** (Strict monotonicity on the number of words)**.** *Let $L_1$ and $L_2$ be two regular languages such that $\forall\, k \in \mathbb{N} : |C_k(L_1)| \leq |C_k(L_2)|$ and $\exists\, k_0 \in \mathbb{N} : |C_{k_0}(L_1)| < |C_{k_0}(L_2)|$. Then, $ent\bullet(L_1) < ent\bullet(L_2)$.*

*Proof.* Language $L_1$ does not contain the maximum number of sequences of length $k_0$, because $|C_{k_0}(L_1)| < |C_{k_0}(L_2)|$. Let $l$ be a sequence such that $l \in \Sigma^*$, $|l| = k_0$ and $l \notin L_1$. Let us consider language $L_1' = L_1 \cup \{l\}$. Then, $ent\bullet(L_1) < ent\bullet(L_1')$, because of the monotonicity of the short-circuit entropy measure, see [8]. According to Theorem 1, $ent\bullet(L_1') \leq ent\bullet(L_2)$. Thus, $ent\bullet(L_1) < ent\bullet(L_2)$. □

Theorems 1 and 2 allow comparing short-circuit entropy measures of languages that are not in the containment relationship. According to Theorem 2, for example, for two languages $L_1 = \{\langle a, a\rangle, \langle a, a, a\rangle, \langle a, a, a, a\rangle, ...\}$ and $L_2 = \{\langle b\rangle, \langle b, b\rangle, \langle b, b, b\rangle, \langle b, b, b, b\rangle, ...\}$ it holds that $ent\bullet(L_1) < ent\bullet(L_2)$.

### C. Precision and Recall

Let $M$ and $L$ be two regular languages that capture the traces of the model and log, respectively. The intersection of $M$ and $L$ is a regular language, refer to [13]. One can use $ent\bullet(M \cap L)$ to estimate the cardinality of the collection of all the traces shared by the model and log. Consequently, we define the entropy-based precision ($prec$) and recall ($recall$) between $M$ and $L$ as follows:

$$prec(M, L) = \frac{ent\bullet(M \cap L)}{ent\bullet(M)}, \qquad recall(M, L) = \frac{ent\bullet(M \cap L)}{ent\bullet(L)}.$$

In [8], we showed that as the number of traces shared by the model and log increases, the entropy-based precision and recall also increase. Next, we demonstrate three additional properties of the entropy-based precision and recall, which also hold for their extensions proposed in Section V.

**Theorem 3** (Monotonicity)**.** *Let $L_1$ and $L_2$ be two event logs (let $M_1$ and $M_2$ be two regular languages) such that $L_1 \subset L_2$ ($M_1 \subset M_2$). Let $M$ be a regular language (let $L$ be an event log). Then, $prec(M, L_1) \leq prec(M, L_2)$ ($recall(M_1, L) \leq recall(M_2, L)$).*

*Proof.* Since it holds that $L_1 \subset L_2$ ($M_1 \subset M_2$), it also holds that $M \cap L_1 \subseteq M \cap L_2$ ($M_1 \cap L \subseteq M_2 \cap L$). Because of the monotonicity of the short-circuit entropy, it holds that $ent\bullet(M \cap L_1) \leq ent\bullet(M \cap L_2)$ ($ent\bullet(M_1 \cap L) \leq ent\bullet(M_2 \cap L)$). Hence, it also holds that $prec(M, L_1) \leq prec(M, L_2)$ ($recall(M_1, L) \leq recall(M_2, L)$). □

Theorem 3 shows that the "monotonicity of languages" implies the monotonicity of the corresponding precision and recall values. Next, we demonstrate two additional properties of precision and recall that follow from Theorems 1 and 2. These properties are discussed and exemplified in Section V.

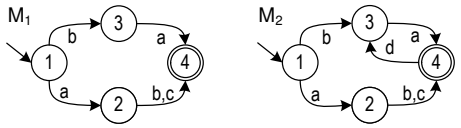
**Theorem 4** (Generalized monotonicity)**.** *Let $L_1$ and $L_2$ be two event logs (let $M_1$ and $M_2$ be two regular languages) and let $M$ be a regular language (let $L$ be an event log) such that $\forall k \in \mathbb{N} : |C_k(M \cap L_1)| \leq |C_k(M \cap L_2)$ ($\forall k \in \mathbb{N} : |C_k(M_1 \cap L)| \leq |C_k(M_2 \cap L)|$). Then, $prec(M, L_1) \leq prec(M, L_2)$ ($recall(M_1, L) \leq recall(M_2, L)$).*

Theorem 4 follows from Theorem 1 applied to $M \cap L_1$ and $M \cap L_2$ ($M_1 \cap L$ and $M_2 \cap L$). This result can be conveniently refined into the next one.

**Theorem 5** (Generalized strict monotonicity)**.** *Let $L_1$ and $L_2$ be two event logs (let $M_1$ and $M_2$ be two regular languages) and let $M$ be a regular language (let $L$ be an event log) such that $\forall\, k \in \mathbb{N} : |C_k(M \cap L_1)| \leq |C_k(M \cap L_2)|$ ($\forall\, k \in \mathbb{N} : |C_k(M_1 \cap L)| \leq |C_k(M_2 \cap L)|$) and $\exists\, k_0 \in \mathbb{N} : |C_{k_0}(M \cap L_1)| < |C_{k_0}(M \cap L_2)|$ ($\exists\, k_0 \in \mathbb{N} : |C_{k_0}(M_1 \cap L)| < |C_{k_0}(M_2 \cap L)|$). Then, $prec(M, L_1) < prec(M, L_2)$ ($recall(M_1, L) < recall(M_2, L)$).*

Theorem 5 follows immediately from the application of Theorem 2 to languages $M \cap L_1$ and $M \cap L_2$ ($M_1 \cap L$ and $M_2 \cap L$).

Consider the two DFAs shown in Figure 3a and Figure 3b that recognize languages $M_1$ and $M_2$, respectively, and three simple event logs $L_1 = \{\langle a, b\rangle, \langle b, a\rangle\}$, $L_2 = \{\langle\rangle, \langle a, b\rangle, \langle b, a\rangle, \langle a, c\rangle, \langle b, a, b\rangle\}$, and $L_3 = \{\langle b, a, d\rangle, \langle b, a, b\rangle\}$.




(a) (b)

Figure 3: Two finite automata.

The entropy-based precision and recall values for all the combinations of these two regular languages and three event logs are listed in Table I below.

Table I: The entropy-based precision and recall values.

| Model | Log | *prec* | *recall* |
|---|---|---|---|
| $M_1$ | $L_1$ | 0.874 | 1.000 |
| $M_1$ | $L_2$ | 1.000 | 0.745 |
| $M_1$ | $L_3$ | 0.000 | 0.000 |
| $M_2$ | $L_1$ | 0.754 | 1.000 |
| $M_2$ | $L_2$ | 0.863 | 0.745 |
| $M_2$ | $L_3$ | 0.000 | 0.000 |

The values in Table I justify, e.g., that $M_1$ contains all the traces from $L_1$ ($recall(M_1, L_1) = 1$) and does not contain some traces from $L_2$ ($recall(M_1, L_2) < 1$). One

can also conclude that $L_2$ "covers" all the behavior of $M_1$ ($prec(M_1, L_2) = 1$). Furthermore, $M_2$ does not contain some traces from $L_2$ ($recall(M_2, L_2) < 1$). Interestingly, according to the results in [8], it holds that $prec(M_2, L_1) < prec(M_1, L_1)$ and $prec(M_2, L_2) < prec(M_1, L_2)$, because $L_1 \subset M_1 \subset M_2$, $M_1 \cap L_2 = M_2 \cap L_2$, and $(M_1 \cap L_2) \subset M_1$.

Note that $L_3$ does not intersect with $M_1$ (or $M_2$). Consequently, despite some shared subsequences in their traces, e.g., $\langle b, a \rangle$, the corresponding precision and recall values equal to zero. This limitation is addressed in the next section.

## V. Partially Matching Approach

In this section, we extend the conformance checking approach summarized in the previous section with support for partial matching of the compared model and log. We first define additional notions and discuss their properties, refer to Section V-A. We then use these notions to propose new precision and recall measures, refer to Section V-B.

### A. Entropy and $\tau$-closure of Regular Languages

Let $B = (Q, \Lambda, \delta, q_0, A)$ be a DFA that recognizes language $L$, i.e., $lang(B) = L$. We construct the $\tau$-closure of $B$, denoted by $B'$, as follows: $B' = (Q, \Lambda, \delta', q_0, A)$, $\delta'(q_1, a) = q_2$ iff $(\delta(q_1, a) = q_2) \vee ((a = \tau) \wedge (\exists\, a' : \delta(q_1, a') = q_2))$. In other words, for each two states connected via a transition in $B$, we add an additional silent transition that connects these states. We call $B'$ ($lang(B')$) the *$\tau$-closure* of $B$ ($L$). Figure 4 shows the $\tau$-closure of the automaton from Figure 2a.

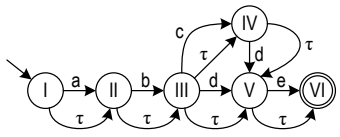



Figure 4: $\tau$-closure of the DFA from Figure 2a.

By $L'$, we denote the language recognized by $B'$. Note that $L \subseteq L'$. Next, we state two important properties of the short-circuit entropy over the $\tau$-closures of regular languages.

**Theorem 6** (Monotonicity of $\tau$-closure)**.**
*Let $L_1$ and $L_2$ be two regular languages such that $L_1 \subset L_2$. Then, it holds that $ent\bullet(L_1') \leq ent\bullet(L_2')$.*

*Proof.* For each $\alpha \in L_1$ it holds that $\alpha \in L_2$. Consequently, all the sequences obtained from $\alpha$ by the $\tau$-closure operation belong to both $L_1'$ and $L_2'$. This implies $L_1' \subseteq L_2'$. Consequently, $ent\bullet(L_1') \leq ent\bullet(L_2')$ because of the monotonicity of the $ent\bullet$ measure. □

Hence, $\tau$-closure relaxes the strict monotonicity. Note that in case the $\tau$-closure operation does not preserve the strict monotonicity, both measures, the exact trace matching and the partial matching, can be applied to obtain the complete information about deviations between the languages. We now define a condition under which one obtains strict monotonicity.

**Theorem 7** (Strict monotonicity of $\tau$-closure)**.**
*Let $L_1$ and $L_2$ be two regular languages, such that $\exists\, \alpha \in L_2 : \alpha \notin L_1'$ and $\forall\, \beta \in L_1 : \beta \in L_2'$, then $L_1' \subset L_2'$ and $ent\bullet(L_1') < ent\bullet(L_2')$.*

*Proof.* Since for each $\beta \in L_1$ it holds that $\beta \in L_2'$, it also holds that $\{\beta\}' \subseteq L_2'$, because if a $\tau$-closure of a language contains $\beta$, it contains all the sequences obtained from $\beta$ using the $\tau$-closure operation. Consequently, $L_1' \subseteq L_2'$. Additionally, since $\exists\, \alpha \in L_2 : \alpha \notin L_1'$, it is a strict inclusion $L_1' \subset L_2'$, and $ent\bullet(L_1') < ent\bullet(L_2')$. □

Suppose that $L_1$ and $L_2$ are two regular languages, such that $L_1' = (L_2')\backslash\{\alpha\}$ and $\alpha \in L_2$, i.e., their $\tau$-closures differ in one sequence $\alpha$. Both $L_1'$ and $L_2'$ contain all "sub-words" of $\alpha$, i.e., $\{\alpha\}'\backslash\alpha \subseteq L_1'$ and $\{\alpha\}'\backslash\alpha \subset L_2'$. According to Theorem 7, it holds that $ent\bullet(L_1') < ent\bullet(L_2')$. Consequently, one may note that a "long" sequence $\alpha$ belonging to $L_2$ which cannot be generated by constructing a $\tau$-closure of $L_1$ plays a role when comparing $ent\bullet(L_1')$ and $ent\bullet(L_2')$, even if $L_1'$ and $L_2'$ share the same set of its "sub-words".

Consider two models that recognize languages $M_1 = \{\langle a, b \rangle, \langle a, c \rangle \langle b, c \rangle \langle d \rangle\}$ and $M_2 = \{\langle a, b, c \rangle, \langle d \rangle\}$. Obviously, $M_2' = M_1' \cup \{\langle a, b, c \rangle\}$. Let $L = \{\langle a, b, c \rangle\}$ be a language. According to Theorem 7, $recall(M_2', L') > recall(M_1', L')$, as the "long" sequence $\langle a, b, c \rangle$ belongs to $M_2$ and cannot be obtained by applying the $\tau$-closure operation to $M_1$.

### B. Precision and Recall

Let $M$ and $L$ be languages recognized by DFAs that encode a model and log, respectively. The relations between $M$ and $L$ and their $\tau$-closures are represented in Figure 5.

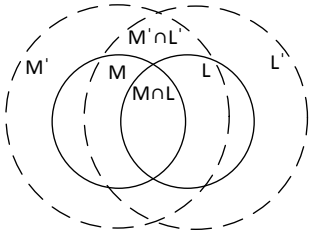



Figure 5: Intersection of two languages and their $\tau$-closures.

We propose to measure precision and recall between the model and log based on the $\tau$-closures of $M$ and $L$ as follows:

$$prec_\tau(M, L) = \frac{ent\bullet(M' \cap L')}{ent\bullet(M')},$$

$$recall_\tau(M, L) = \frac{ent\bullet(M' \cap L')}{ent\bullet(L')},$$

where $M'$ and $L'$ are the $\tau$-closures of $M$ and $L$, respectively. Note that first $M'$, $L'$, and $M' \cap L'$ are constructed and only after that the short-circuit entropy is computed. Table II extends Table I by also showing the new precision and recall values for the corresponding models and logs.

These example values show that although log $L_3$ has no common traces with the models, they can be partially matched. According to Theorem 7, it holds that $L_1' \subset L_2'$, and according to Theorem 3 (applied to $L_1'$, $L_2'$, $M_1'$, and $M_2'$; note that

Table II: The entropy-based precision and recall values, both original and based on the $\tau$-closures of languages.

| Model | Log | $prec$ | $recall$ | $prec_\tau$ | $recall_\tau$ |
|---|---|---|---|---|---|
| $M_1$ | $L_1$ | 0.874 | 1.000 | 0.873 | 1.000 |
| $M_1$ | $L_2$ | 1.000 | 0.745 | 1.000 | 0.960 |
| $M_1$ | $L_3$ | 0.000 | 0.000 | 0.873 | 0.811 |
| $M_2$ | $L_1$ | 0.754 | 1.000 | 0.615 | 1.000 |
| $M_2$ | $L_2$ | 0.863 | 0.745 | 0.704 | 0.960 |
| $M_2$ | $L_3$ | 0.000 | 0.000 | 0.733 | 0.966 |

$\tau$-closures of regular languages are also regular languages), new precision values for $L_1$ (0.873 and 0.615) are indeed less than or equal to the corresponding new precision values for $L_2$ (1.000 and 0.704). Interestingly, $prec_\tau(M_1, L_2) = 1$ because $M_1' \subseteq L_2'$. Also, note that the absolute values of the reported measures are of minor importance, as those are their relations that provide useful insights. According to Theorem 7, $L_1' \subset L_3'$, similarly, the new precision values for $L_3$ (0.873 and 0.733) are greater than or equal to the corresponding new precision values for $L_1$ (0.873 and 0.615).

It is easy to verify that $M_1' \subset M_2'$, refer to Figure 3. Then, according to Theorem 3, recall values for $M_2$ are always greater or equal to the corresponding recall values for $M_1$.

Let us take a closer look at logs $L_2$ and $L_3$. None of these logs includes the other; same holds for logs $L_2'$ and $L_3'$. It holds that $M_2' \cap L_2' = \{\langle\rangle, \langle a\rangle, \langle b\rangle, \langle c\rangle, \langle a,b\rangle, \langle b,a\rangle, \langle a,c\rangle\}$ and $M_2' \cap L_3' = \{\langle\rangle, \langle a\rangle, \langle b\rangle, \langle d\rangle, \langle a,b\rangle, \langle b,a\rangle, \langle a,d\rangle, \langle b,d\rangle, \langle b,a,d\rangle\}$. The former language contains less number of sequences of the length two and three. Hence, by Theorem 5, it must hold that $prec_\tau(M_2, L_2) < prec_\tau(M_2, L_3)$. Indeed, according to the values in Table II: $prec_\tau(M_2, L_2) = 0.704$ and $prec_\tau(M_2, L_3) = 0.733$. Note that this property allows comparing languages over different alphabets.

## VI. Experimental Results

The proposed partial matching technique has been implemented and is publicly available.[1] Next, we evaluate this implementation over synthetic and real-life datasets.

### A. Synthetic Dataset

In this section, we experiment with a synthetic event log and a set of corresponding process models described in [14], [15]. The event log is defined as this set of traces: $L = \{\langle A,B,D,E,I\rangle, \langle A,C,D,G,H,F,I\rangle, \langle A,C,G,D,H,F,I\rangle, \langle A,C,H,D,F,I\rangle, \langle A,C,D,H,F,I\rangle\}$. The set of process models consists of nine Petri nets shown in Figure 6. The reachability graphs of all these nine Petri nets can be encoded as DFAs. Table III shows precision and recall values between languages recognized by these DFAs and $L$. All these values were computed in close to real-time (in milliseconds) using Intel Core i3-3110M CPU @2.40 GHz with 4 GB RAM.

The values in the left-most $recall$ column in Table III measure the share of traces from the log that are also the traces of the model. Six models can "replay" all the log traces, hence the recall values of one. The *Single trace* model accepts only one trace from the log, which leads to the recall value of less than one. The *All parallel* model, which imposes a restriction that all the labels must appear in a trace, and the *Round robin* model, which executes all the activities in a particular order without skipping them, do not accept a single trace from the log. Therefore, the corresponding recall values equal to zero. Note that the recall values that are based on the $\tau$-closures of languages show that the *All parallel* and *Round robin* model describe traces that preserve the order (assuming skips in the model traces) of events as they appear in the traces of the event log, see the corresponding recall values of 1.000.

[1] https://github.com/akalenkova/eigen-measure.

Table III: Precision and recall values for synthetic log.

| Model | $prec$ | $recall$ | $prec_\tau$ | $recall_\tau$ |
|---|---|---|---|---|
| $Original\ model$ | 0.979 | 1.000 | 0.998 | 1.000 |
| $Single\ trace$ | 1.000 | 0.798 | 1.000 | 0.732 |
| $Separated\ traces$ | 1.000 | 1.000 | 1.000 | 1.000 |
| $Flower\ model$ | 0.125 | 1.000 | 0.479 | 1.000 |
| $H\ and\ G\ in\ parallel$ | 0.889 | 1.000 | 0.986 | 1.000 |
| $H\ and\ G\ in\ loops$ | 0.568 | 1.000 | 0.933 | 1.000 |
| $D\ in\ a\ loop$ | 0.758 | 1.000 | 0.970 | 1.000 |
| $All\ parallel$ | 0.000 | 0.000 | 0.656 | 1.000 |
| $Round\ robin$ | 0.000 | 0.000 | 0.479 | 1.000 |

Let us now examine the precision values. We ranked the models in accordance with their precision values in Table IV.

The table shows our rankings (last two columns) and compares them with the rankings of other conformance techniques calculated for the same models and log in [14], [15]. Concretely, these techniques are used (refer to columns 2–8 in the table): *Set difference* (*SD*) [16], *Negative events* (*NE*) [17], *Escaping edges* (*ETC*) [18], *Alignment-based ETC precision* (*ETCa*) [19], *Projected conformance checking* (*PCC*) [20], *Anti-alignment precision* (*AA*) [14], [21], and *k-order Markovian abstractions* (*MAP*$^k$) [15]. Greater values of $k$ for the latter technique correspond to less abstraction in the encodings of models and logs and, thus, more "precise" precision measurements. Column *EB* stands for the *entropy-based* approach from Section IV, while *EB*$^\tau$ for its extension from Section V.

The *entropy-based* approach (*EB*) ranks the models quantifying the behavior which is "covered" by the traces of the event log. Because models *All parallel* and *Round robin* are not covered at all, their precision w.r.t. the log is the least, while *Single trace* and *Separated traces* models are totally covered and, thus, they have the best precision w.r.t. the log.

The ranking produced by the partial matching approach (*EB*$^\tau$) is closest to that one by *anti-alignment precision* (*AA*). The difference is that both our approaches show that model *H and G in loops* has a lower precision than the model *D in a loop*. Indeed, these models share the same traces with the log, but the behavior described in *H and G in loops* model has more variability, i.e., the number of sequences between $C$ and $F$ is higher for *H and G in loops* model. The monotonicity of the approach reported in [14], [21] is based on finite concepts, such as maximal length of the log trace, while we propose a more general approach capable of assessing infinite behaviors described in models. Finally, note that in contrast to the anti-alignment precision, using both our precision measures, i.e., *EB* and *EB*$^\tau$, allows for a more complete analysis by differentiating between *Flower* model and *Round robin* model.

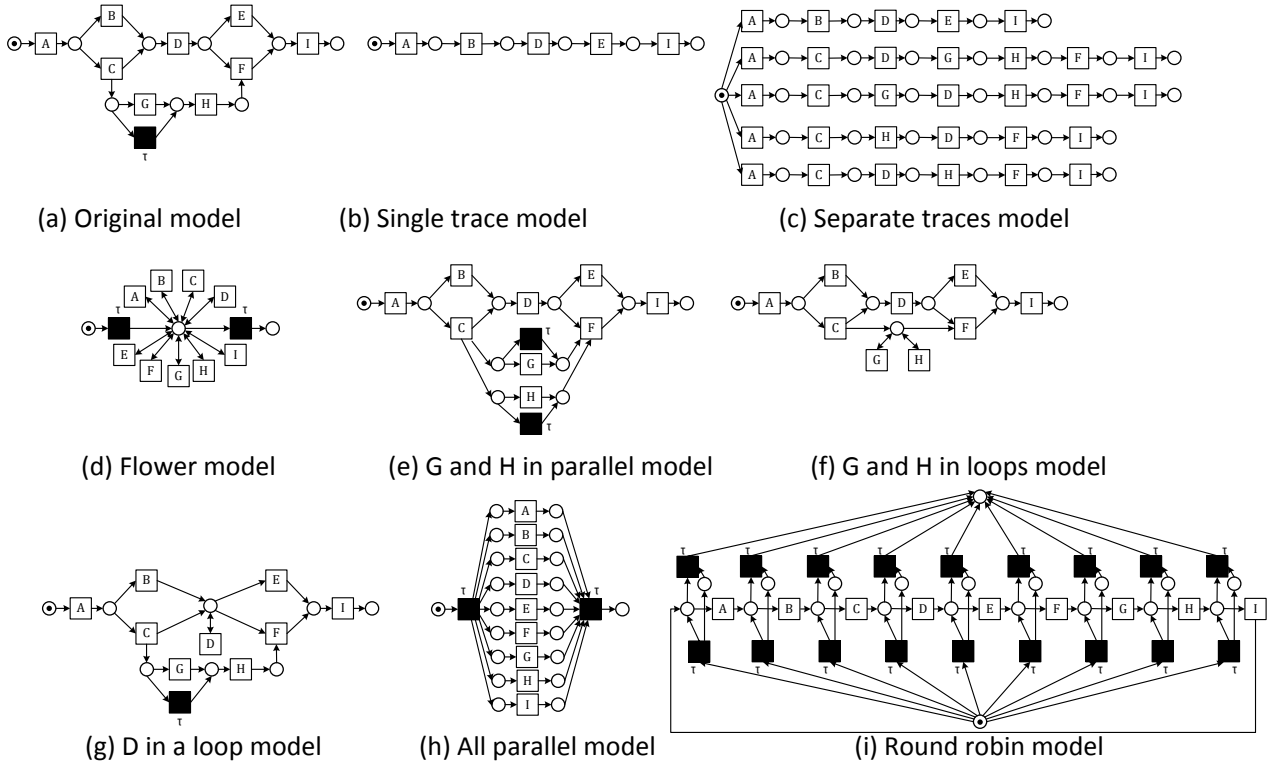



Figure 6: Artificial process models from [14], [15].

Table IV: Rankings of precision values for synthetic log.

| **Model** | $SD$ | $ETC_a$ | $NE$ | $PCC$ | $AA$ | $MAP^1$ | $MAP^{2\text{-}7}$ | $EB$ | $EB^\tau$ |
|---|---|---|---|---|---|---|---|---|---|
| $Original$ | 7 | 7 | 9 | 8 | 7 | 7 | 7 | 7 | 7 |
| $Single\ trace$ | 8 | 8 | 6 | 8 | 8 | 7 | 8 | 8 | 8 |
| $Separated\ traces$ | 8 | 8 | 8 | 7 | 8 | 7 | 8 | 8 | 8 |
| $Flower$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 3 | 1 |
| $H\ and\ G\ in\ parallel$ | 6 | 3 | 7 | 6 | 6 | 5 | 6 | 6 | 6 |
| $H\ and\ G\ in\ loops$ | 1 | 5 | 5 | 4 | 5 | 3 | 3 | 4 | 4 |
| $D\ in\ a\ loop$ | 1 | 6 | 4 | 5 | 4 | 5 | 4 | 5 | 5 |
| $All\ parallel$ | 1 | 2 | 2 | 2 | 3 | 2 | 2 | 1 | 3 |
| $Round\ robin$ | 1 | 4 | 3 | 3 | 1 | 4 | 5 | 1 | 1 |

### *B. Real-life Event Data*

Next, we investigate the scalability of our approach to verify whether it can be applied to real-life event logs. We have analyzed BPI Challenge (BPIC) event logs[2], which are publicly available logs of real-life IT systems, and an event log of a booking flight system (BFS). Prior to the analysis, we filtered out infrequent events that appear less than in 80% of traces using *Filter Log using Simple Heuristics* Process Mining Framework (ProM) [22] plugin.[3] For the filtered event logs, the number of traces in a single log varies from 596 to 11,636 and the overall length of all the traces in a single event log varies from 1,403 to 164,144. From each event log, a Petri net was discovered using the Inductive miner [3]. This discovery technique constructs bounded Petri nets, such that their reachability graphs are DFAs. We used these DFAs as representations of model behaviors to apply the partial matching technique described in this paper. Since the entropy-based approach is applicable to $\tau$-free DFAs only, the DFAs with silent transitions were converted to equivalent $\tau$-free DFAs. It was crucial to estimate the applicability of our approach in order to show that despite the potential state space explosion, it is still computationally feasible to construct an equivalent $\tau$-free DFA from an initial DFA with silent transitions [13] obtained via the $\tau$-closure operation. In our experiments, the number of automaton states increases by no more than 10 times. The results of experiments, including the sizes of the automatons, time (in seconds) taken for the determinization of automata, entropy and precision and recall values calculation, are presented in Table V. In these experiments, we used Intel Core(TM) i7-3970X CPU @3.50 GHz with 64 GB RAM.

[2]BPIC logs: https://data.4tu.nl/repository/collection:event_logs_real.

[3]All the filtered logs, including the BFS log, are distributed with the implementation at https://github.com/akalenkova/eigen-measure.

## VII. Conclusion

In this paper, we proposed an extension of our entropy-based precision and recall conformance measures, proposed

Table V: Time of determinization and entropy calculation for real-life event logs (in seconds).

| Event log | Automaton | # States / # Transitions | Determiniz. time | Entropy calc. time | Precision / Recall |
|---|---|---|---|---|---|
| BPIC'12 | $L'$ | 90,557 / 446,847 | 141.455 | 4,641.421 | |
| | $M' \cap L'$ | 90,557 / 446,847 | - | 4,733.990 | 0.709 / |
| | $M'$ | 3 / 33 | 0.001 | 0.013 | 1.000 |
| BPIC'13 closed | $L'$ | 216 / 629 | 0.171 | 0.235 | |
| | $M' \cap L'$ | 216 / 629 | - | 0.661 | 0.960 / |
| | $M'$ | 1 / 3 | 0.001 | 0.010 | 1.000 |
| BPIC'13 incidents | $L'$ | 24,336 / 72,994 | 1,909.794 | 2.187 | |
| | $M' \cap L'$ | 24,336 / 72,994 | - | 1.552 | 0.995 / |
| | $M'$ | 1 / 3 | 0.001 | 0.011 | 1.000 |
| BPIC'13 open | $L'$ | 17 / 31 | 0.012 | 0.123 | |
| | $M' \cap L'$ | 17 / 31 | - | 0.003 | 0.980 / |
| | $M'$ | 1 / 2 | 0.000 | 0.011 | 1.000 |
| BFS'13 | $L'$ | 22,359 / 200,254 | 37.947 | 2.427 | |
| | $M' \cap L'$ | 7,542 / 45,163 | - | 1.516 | 0.939 / |
| | $M'$ | 514 / 3,340 | 1.625 | 0.153 | 0.825 |

in [8], which quantify the similarity between traces described in a designed process model and its corresponding executed traces recorded in an event log. While precision quantifies how well the traces of the model are represented in the log, recall measures how well the traces in the log are represented in the model. The extension addresses the phenomenon of partially matching traces, i.e., non-identical traces that nevertheless describe similar sequences of process steps. The extended measures exhibit the intuitively desired property which establishes that more partially matching traces with matches of greater length in the compared model and log lead to greater precision and recall values. In [8], we showed that the entropy-based precision and recall fulfill a range of important properties of conformance measures that are not fulfilled by the state of the art measures. The extended measures inherit all those properties of the original measures, while all the properties of the extended measures demonstrated in this paper also apply to the original measures.

The reported qualitative and quantitative evaluation of the proposed measures suggests that they can be useful in industrial settings. We acknowledge that the extended measures, still, have several limitations, which naturally give rise to future work. Firstly, the proposed measures do not support systems which cannot be described as DFAs, for example infinite-state systems. To address this limitation, coverability graphs may yield useful. The proposed extension based on the dilution of the model and log traces has proven effective, but also demonstrated a significant negative impact on the efficiency of the measures. To cope with this deficiency, secondly, we plan to exploit the properties of topological entropy and structural features of log automata to simplify certain "expensive", in terms of time, computations. Thirdly, the proposed approach considers distinct traces only and can be extended to take into account frequencies of traces in event logs. Finally, even though the state of the art conformance measures do not satisfy the properties for completely matching traces, refer to [6], [10] for details, they may satisfy the properties for partially matching traces. Therefore, one can verify if other measures satisfy the properties put forward in this work. Also, we envision that the discussion on the desired properties for conformance measures will continue within the process mining community.

## ACKNOWLEDGMENT

Artem Polyvyanyy was partly supported by the Australian Research Council Discovery Project DP180102839. Anna Kalenkova was supported by the Basic Research Program at the Higher School of Economics.

## REFERENCES

[1] W. van der Aalst, *Process Mining: Data Science in Action*, 2016.
[2] W. van der Aalst, A. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128–1142, 2004.
[3] S. Leemans, D. Fahland, and W. van der Aalst, "Discovering Block-Structured Process Models from Incomplete Event Logs," in *ATPN'2014*, ser. LNCS. Springer, 2014, vol. 8489, pp. 91–110.
[4] A. Augusto, R. Conforti, M. Dumas, M. La Rosa, and A. Polyvyanyy, "Split miner: automated discovery of accurate and simple business process models from event logs," *KAIS*, pp. 1–34, 2018.
[5] W. van der Aalst, A. Adriansyah, and B. van Dongen, "Replaying history on process models for conformance checking and performance analysis," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 182–192, 2012.
[6] A. Polyvyanyy, W. van der Aalst, A. ter Hofstede, and M. Wynn, "Impact-driven process model repair," *ACM Transactions on Software Engineering and Methodology*, vol. 25, no. 4, pp. 1–60, 2017.
[7] J. Carmona, B. van Dongen, A. Solti, and M. Weidlich, *Conformance Checking—Relating Processes and Models*. Springer, 2018.
[8] A. Polyvyanyy, A. Solti, M. Weidlich, C. Di Ciccio, and J. Mendling, "Monotone precision and recall measures for comparing executions and specifications of dynamic systems," *CoRR*, vol. abs/1812.07334, 2018.
[9] W. Frakes and R. Baeza-Yates, *Information Retrieval: Data Structures and Algorithms*. NJ, USA: Prentice-Hall, Inc., 1992.
[10] N. Tax, X. Lu, N. Sidorova, D. Fahland, and W. van der Aalst, "The imprecisions of precision measures in process mining," *Information Processing Letters*, vol. 135, pp. 1–8, 2018.
[11] W. van der Aalst, "Relating process models and event logs—21 conformance propositions," in *Proceedings of ATAED satellite event for ATPN 2018*, ser. CEUR Workshop Proceedings, vol. 2115, 2018, pp. 56–74.
[12] T. Ceccherini-Silberstein, A. Machì, and F. Scarabotti, "On the entropy of regular languages," *Theor. Comp. Sci.*, vol. 307, pp. 93–102, 2003.
[13] J. Hopcroft, R. Motwani, and J. Ullman, *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*, Boston, USA, 2006.
[14] B. van Dongen, J. Carmona, and T. Chatain, "A unified approach for measuring precision and generalization based on anti-alignments," in *Business Process Management*. Cham: Springer, 2016, pp. 39–56.
[15] A. Augusto, A. Armas-Cervantes, R. Conforti, M. Dumas, M. La Rosa, and D. Reissner, "Abstract-and-compare: A family of scalable precision measures for automated process discovery," in *Business Process Management*. Cham: Springer International Publishing, 2018, pp. 158–175.
[16] G. Greco, A. Guzzo, L. Pontieri, and D. Sacca, "Discovering expressive process models by clustering log traces," *IEEE Trans. on Knowl. and Data Eng.*, vol. 18, no. 8, pp. 1010–1027, 2006.
[17] J. De Weerdt, M. De Backer, J. Vanthienen, and B. Baesens, "A robust f-measure for evaluating discovered process models," in *CIDM*. IEEE, 2011, pp. 148–155.
[18] J. Muñoz-Gama and J. Carmona, "A fresh look at precision in process conformance," in *Business Process Management*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 211–226.
[19] A. Adriansyah, J. Munoz-Gama, J. Carmona, B. van Dongen, and W. van der Aalst, "Measuring precision of modeled behavior," *Inf. Syst. and e-Business Management*, vol. 13, no. 1, pp. 37–67, 2015.
[20] S. Leemans, D. Fahland, and W. van der Aalst, "Scalable process discovery and conformance checking," *Software & Systems Modeling*, vol. 17, no. 2, pp. 599–631, 2018.
[21] T. Chatain and J. Carmona, "Anti-alignments in conformance checking – the dark side of process models," in *ATPN'2016*, pp. 240–258.
[22] B. van Dongen, A. de Medeiros, H. Verbeek, A. Weijters, and W. van der Aalst, "The ProM Framework: A new era in process mining tool support," in *ATPN'2005*, pp. 444–454.

# Mining Uncertain Event Data in Process Mining

Marco Pegoraro and Wil M.P. van der Aalst
Process And Data Science research group (PADS)
Department of Computer Science, RWTH Aachen University
Aachen, Germany
Email: {pegoraro,wvdaalst}@pads.rwth-aachen.de

***Abstract*—Nowadays, more and more process data are automatically recorded by information systems, and made available in the form of *event logs*. Process mining techniques enable process-centric analysis of data, including automatically discovering process models and checking if event data conform to a certain model. In this paper we analyze the previously unexplored setting of uncertain event logs: logs where quantified uncertainty is recorded together with the corresponding data. We define a taxonomy of uncertain event logs and models, and we examine the challenges that uncertainty poses on process discovery and conformance checking. Finally, we show how upper and lower bounds for conformance can be obtained aligning an uncertain trace onto a regular process model.**

## I. Introduction

Over the last decades, the concept of *process* has become more and more central in formally describing the activities of businesses, companies and other similar entities, structured in specific steps and phases. A process is thus defined as a well-structured set of activities, possibly performed by multiple actors (*resources*), which contribute to the completion of a specific task or to the achievement of a specific goal.

The processes that govern the innards of business companies are increasingly supported by software tools. Performing specific activities is both aided and recorded by *process-aware information systems* (PAISs), which support the definition and management of processes. The information regarding the execution of processes, which includes time, case and activity information, can then be extracted from PAISs in the form of an *event log*, a database or file containing the digital trace of the operations carried out in the context of the execution of a process and recorded as *events*. The discipline of *process mining* concerns the automatic analysis of event logs, with the goal of extracting knowledge regarding e.g. the structure of the process, the conformity of events to a specific normative process model, the performances in executing the process, the relationships between groups of actors in the process.

In this paper we will consider the analysis of a specific class of event logs: the logs that contain *uncertain event data*. Uncertain events are recordings of executions of specific activities in a process which are enclosed with an indication of uncertainty in the event attributes. Specifically, we consider the case where the attributes of an event are not recorded as a precise value but as a range or a set of alternatives.

The recording of uncertain event data is a common occurrence in process management. The *Process Mining Manifesto* [1] describes a fundamental property of event data as *trustworthiness*, the assumption that the recorded data can be considered correct and accurate. In a general sense, uncertainty as defined here is an explicit absence of trustworthiness, with an indication of uncertainty recorded together with the event data. In the taxonomy of event data proposed in the Manifesto the logs at the two lower levels of quality frequently lack trustworthiness, and thus can be uncertain. This encompasses a wide range of processes, such as event logs of document and product management systems, error logs of embedded systems, worksheets of service engineers, and any process recorded totally or partially on paper. There are many possible causes behind the recording of uncertain event data, such as:

- *Incorrectness*. In some instances, the uncertainty is simply given by errors occurred while recording the data itself. Faults of the information system, or human mistakes in a data entry phase can all lead to missing or altered event data that can be subsequently modeled as uncertain event data.
- *Coarseness*. Some information systems have limitations in their way of recording data - often tied to factors like the precision of the data format - such that the event data can be considered uncertain. A typical example is an information system that only records the date, but not the time, of the occurrence of an event: if two events are recorded in the same day, the order of occurrence is lost. This is an especially common circumstance in the processes that are, partially or completely, recorded on paper and then digitalized. Another factor that can lead to uncertainty in the time of recording is the information system being overloaded and, thus, delaying memorization of data. This type of uncertainty can also be generated by the limited sensibility of a sensor.
- *Ambiguity*. In some cases, the data recorded is not an identifier of a certain event attribute; in these instances, the data needs to be interpreted, either automatically or manually, in order to obtain a value for the event attribute. Uncertainty can arise if the meaning of the data is ambiguous and cannot be interpreted with precision. Example are data in the form of images, text, or video.

Aside from the causes, we can individuate other types of uncertain event logs based on the frequency of uncertain data. Uncertainty can be *infrequent*, when a specific attribute is only seldomly recorded together with explicit uncertainty; the uncertainty is rare enough that uncertain events can be

978-1-7281-0919-0/19/$31.00 ©2019 IEEE
DOI 10.1109/ICPM.2019.00023

considered outliers. Conversely, *frequent* uncertain behavior of the attribute is systematic, pervasive in a high number of traces, and thus not to be considered an outlier. The uncertainty can be considered part of the process itself. These concepts are not meant to be formal, and are laid out to distinguish between logs that are still processable regardless of the uncertainty, and logs where the uncertainty is too invasive to analyze them with existing process mining techniques.

In this paper we propose a taxonomy of the different types of explicit uncertainty in process mining, together with a formal, mathematical formulation. As an example of practical application, we will consider the case of conformance checking [2], and we will apply it to uncertain data by assessing what are the upper and lower bounds on the conformance score for possible values of the attributes in an uncertain trace.

The rest of this paper is organized as follows. Section II discusses previous and related work in the management of uncertain data. Section III proposes a taxonomy of the different possible types of uncertain process data. Section IV contains the formal definitions needed to manage uncertainty. Section V describes a practical application of process mining over uncertain event data, the case of conformance checking through alignments. Section VI shows experimental results on computing conformance checking scores for uncertain data. Finally, Section VII concludes the paper and discusses about future work.

## II. Related Work

As mentioned, the occurrence of data containing uncertainty - in a broad sense - is common both in more classic disciplines like statistics and Data Mining [3] and in process mining [1]; and logs that show an explicit uncertainty in the control flow perspective can be classified in the lower levels of the quality ranking proposed in the process mining manifesto.

Within process mining there exist various techniques to deal with a kind of uncertainty different from the one that we analyze here: missing or incorrect data. This can be considered as a form of non-explicit uncertainty: no measure or indication on the nature of the uncertainty is given in the event log. The work of Suriadi et al. [4] provides a taxonomy of this type of issues in event logs, laying out a series of data patterns that model errors in process data. In these cases, and if this behavior is infrequent enough to allow the event log to remain meaningful, the most common way for existing process mining techniques to deal with missing data is by filtering out the affected traces and performing discovery and conformance checking on the resulting filtered event log. While filtering out missing values is straightforward, various methodologies of event log filtering have been proposed in the past to solve the problem of incorrect event attributes: the filtering can take place thanks to a reference model, which can be given as process specification [5], or from information discovered from the frequent and well-formed traces of the same event log; for example extracting an automaton from the frequent traces [6], computing conditional probabilities of frequent sequences of activities [7], or discovering a probabilistic automaton [8]. In the latter cases, the noise is identified as infrequent behavior.

Some previous work attempt to repair the incorrect values in an event log. Conforti et al. [9] propose an approach for the restoration of incorrect timestamps based on a log automaton, that repairs the total ordering of events in a trace based on correct frequent behavior. Fani Sani et al. [10] define outlier behavior as the unexpected occurrence of an event, the absence of an event that is supposed to happen, and the incorrect order of events in the trace; then, they propose a repairing method based on probabilistic analysis of the context of an outlier (events preceding or following the anomalous event). Again, both of these methods define anomalous/incorrect behavior on the basis of the frequency of occurrence.

The main driving reasons behind this work is to provide the means to treat uncertainty as a relevant part of a process; thus, we aim not to filter it out but model it. In conclusion, there are two novel aspects regarding uncertain data that we intend to address in this work. The first is the *explicitness of uncertainty*: we work with the underlying assumption that the actual value of the uncertain attribute, while not directly provided, is described formally. This is the case when meta-information about the uncertainty in the attribute is available, either deduced from the features of the information system(s) that record the logs or included in the event log itself. Note that, as opposed to all previous work on the topic, the fact that uncertainty is explicit in the data means that the concept of uncertain behavior is completely separated from the concept of infrequent behavior. The second is the goal of *modeling uncertainty*: we consider uncertainty part of the process. Instead of filtering or cleaning the log we introduce the uncertainty perspective in process mining by extending the currently available techniques to incorporate it.

## III. A Taxonomy of Uncertain Event Data

The goal of this section of the paper is to propose a categorization of the different types of uncertainty that can appear in process mining. In process management, a central concept is the distinction between the data perspective (the event log) and the behavioral perspective (the process model). The first one is a static representation of process instances, the second summarizes the behavior of a process. Both can be extended with a concept of explicit uncertainty: this concept also implies an extension of the process mining techniques that have currently been implemented.

In this paper we will focus on uncertainty in event data, while the concept of uncertainty applied to models will be examined in a future work. Specifically, as an example application we will consider computing the conformance score of uncertain process data on classical models.

We can individuate two different notions of uncertainty:

- *Strong uncertainty*: the possible values for the attributes are known, but the probability that the attribute will assume a certain instantiation is unknown or unobservable.
- *Weak uncertainty*: both the possible values of an attribute and their respective probabilities are known.

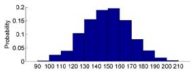
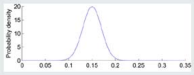
| | Weak uncertainty | Strong uncertainty |
|---|---|---|
| **Discrete data** | Discrete probability distribution  | Set of possible values $\{x, y, z, ...\}$ |
| **Continuous data** | Probability density function  | Interval $\{x \in \mathbb{R} \mid a \leq x \leq b\}$ |

Fig. 1. The four different types of uncertainty.

In the case of a discrete attribute, the strong notion of uncertainty consists on a set of possible values assumed by the attribute. In this case, the probability for each possible value is unknown. Vice-versa, in the weak uncertainty scenario we also have a discrete probability distribution defined on that set of values. In the case of a continuous attribute, the strong notion of uncertainty can be represented with an interval for the variable. Notice that an interval do not indicate a uniform distribution; there is no information on the likelihood of values in it. Vice-versa, in the weak uncertainty scenario we also have a probability density function defined on a certain interval. Figure 1 summarizes this concepts. This leads to very simple representations of explicit uncertainty.

In this paper we consider only the control flow and time perspective of a process – namely, the attributes of the events that allow to discover a process model. These are the unique identifier of a process instance (case ID), the timestamp (often represented by the distance from a fixed origin point, e.g. the *Unix Epoch*), and the activity identifier of an event. Case IDs and activities are values chosen from a finite set of possible values; they are discrete variables. Timestamps, instead, are represented by numbers and thus are continuous variables.

We will also describe an additional type of uncertainty, which lays on the event level rather that the attribute level:

- *Indeterminate event*: there is a chance that the event did not take place even though it was recorded in the event log. Indeterminate events are indicated with a ? symbol, while determinate (regular) events are marked with a ! symbol.

TABLE I
AN EXAMPLE OF STRONGLY UNCERTAIN TRACE.

| Case ID | Timestamp | Activity | Indet. event |
|---|---|---|---|
| {0, 1} | 2011-12-05T00:00 | A | ! |
| 0 | 2011-12-07T00:00 | {B, C, D} | ! |
| 0 | [2011-12-06T00:00, 2011-12-10T00:00] | D | ? |
| 0 | 2011-12-09T00:00 | {A, C} | ! |
| {0, 1, 2} | 2011-12-11T00:00 | E | ? |

Examples of strongly and weakly uncertain traces are shown in Tables I and II respectively.

TABLE II
AN EXAMPLE OF WEAKLY UNCERTAIN TRACE.

| Case ID | Timestamp | Activity | Indet. event |
|---|---|---|---|
| {0:0.9, 1:0.1} | 2011-12-05T00:00 | A | ! |
| 0 | 2011-12-07T00:00 | {B:0.7, C:0.3} | ! |
| 0 | $\mathcal{N}$(2011-12-08T00:00, 2) | D | ?:0.5 |
| 0 | 2011-12-09T00:00 | {A:0.2, C:0.8} | ! |
| {0:0.4, 1:0.6} | 2011-12-11T00:00 | E | ?:0.7 |

## IV. DEFINITIONS

Let us now provide a formal definition of the concept of uncertainty applied to event data.

*Definition 1 (Power Set):* The power set of a set $A$ is the set of all possible subsets of $A$, and is denoted with $\mathcal{P}(A)$. $\mathcal{P}_{NE}(A)$ denotes the set of all the non-empty subsets of $A$: $\mathcal{P}_{NE}(A) = \mathcal{P}(A) \setminus \{\emptyset\}$.

*Definition 2 (Multiset):* A *multiset* is an extension of the concept of set that keeps track of the cardinality of each element. $\mathcal{B}(A)$ is the set of all multisets over some set $A$. Multisets are denoted with square brackets, e.g. $b = [x, x, y]$.

*Definition 3 (Sequence):* Given a set $X$, a finite *sequence* over $X$ of length $n$ is a function $s \in X^* : \{1, \ldots, n\} \to X$, and it is written as $s = \langle s_1, s_2, \ldots, s_n \rangle$. Over the sequence $s$ we define $|s| = n$, $s[i] = s_i$ and $x \in s \Leftrightarrow x \in \text{set}(s)$.

*Definition 4 (Universes):* Let $\mathcal{U}_E$ be the set of all the *event identifiers*. Let $\mathcal{U}_C$ be the set of all the *case id identifiers*. Let $\mathcal{U}_A$ be the set of all the *activity identifiers*. Let $\mathcal{U}_T$ be the totally ordered set of all the *timestamp identifiers*. Let $\mathcal{U}_O = \{!, ?\}$, where the "!" symbol denotes *determined events*, and the "?" symbol denotes *indeterminate events*.

*Definition 5 (Events):* Let us denote with $\mathcal{E}_C = \mathcal{U}_E \times \mathcal{U}_C \times \mathcal{U}_A \times \mathcal{U}_T$ the universe of *certain events*. $\mathcal{E}_{SU} = \mathcal{U}_E \times \mathcal{P}_{NE}(\mathcal{U}_C) \times \mathcal{P}_{NE}(\mathcal{U}_A) \times \mathcal{P}_{NE}(\mathcal{U}_T) \times \mathcal{U}_O$ is the universe of *strongly uncertain events*. $\mathcal{E}_{WU} = \{(e, f) \in \mathcal{U}_E \times (\mathcal{U}_C \times \mathcal{U}_A \times \mathcal{U}_T \not\to [0, 1]) \mid \sum_{(a,c,t) \in dom(f)} f(c, a, t) \leq 1\}$ is the universe of *weakly uncertain events*. Over a strongly uncertain event $(e, c_s, a_s, t_s, u) \in \mathcal{E}_{SU}$ we define the following projection functions: $\pi_c^{\mathcal{E}_{SU}}(e) = c_s$, $\pi_a^{\mathcal{E}_{SU}}(e) = a_s$, $\pi_t^{\mathcal{E}_{SU}}(e) = t_s$ and $\pi_o^{\mathcal{E}_{SU}}(e) = o$.

*Definition 6 (Event logs):* A *certain event log* is a set of events $L_C \subseteq \mathcal{E}_C$ such that every event identifier in $L_C$ is unique. A *strongly uncertain event log* is a set of events $L_{SU} \subseteq \mathcal{E}_{SU}$ such that every event identifier in $L_{SU}$ is unique. A *weakly uncertain event log* is a set of events $L_{WU} \subseteq \mathcal{E}_{WU}$ such that every event identifier in $L_{WU}$ is unique.

A weakly uncertain event log $L_{WU} \subseteq \mathcal{E}_{WU}$ has a corresponding strongly uncertain event log $\overline{L_{WU}} = L_{SU} \subseteq \mathcal{E}_{SU}$ such that $L_{SU} = \{(e, c_s, a_s, t_s, o) \in \mathcal{E}_{SU} \mid \exists_{(e',f) \in L_{WU}}, e = e' \wedge c_s = \{c \in \mathcal{U}_C \mid \exists_{a,t}((c, a, t) \in dom(f) \wedge f(c, a, t) > 0)\} \wedge a_s = \{a \in \mathcal{U}_A \mid \exists_{c,t}((c, a, t) \in dom(f) \wedge f(c, a, t) > 0)\} \wedge t_s = \{t \in \mathcal{U}_T \mid \exists_{c,a}((c, a, t) \in dom(f) \wedge f(c, a, t) > 0)\} \wedge (o = ! \Leftrightarrow (\sum_{(c,a,t) \in dom(f)} f(c, a, t)) = 1) \wedge (o = ? \Leftrightarrow (\sum_{(c,a,t) \in dom(f)} f(c, a, t)) < 1)\}$.

*Definition 7 (Realization of an event log):* $L_C \subseteq \mathcal{E}_C$ is a *realization* of $L_{SU} \subseteq \mathcal{E}_{SU}$ if and only if:

- For all $(e, c, a, t) \in L_C$ there is a distinct $(e', c_s, a_s, t_s, o) \in L_{SU}$ such that $e' = e$, $a \in a_s$, $c \in c_s$ and $t \in t_s$;
- For all $(e, c_s, a_s, t_s, o) \in L_{SU}$ with $o =!$ there is a distinct $(e', c, a, t) \in L_C$ such that $e' = e$, $a \in a_s$, $c \in c_s$ and $t \in t_s$.

$\mathcal{R}_L(L_{SU})$ is the set of all such realizations of the log $L_{SU}$. Note that these definition allow us to transform a weakly uncertain log into a strongly uncertain one, and a strongly uncertain one in a set of certain logs.

## V. Conformance Checking on Uncertain Event Data

As a preliminary application of process mining over uncertain event data we now focus on conformance checking. Starting from an event log and a process model, conformance checking verifies if the event data in the log conforms to the model, providing a diagnostic of the deviations. Conformance checking serves many purposes, such as checking if process instances follow a specific normative model, assessing if a certain execution log has been generated from a specific model, or verifying the quality of a process discovery technique.

The specific scenario we consider in this paper includes:

- Strong uncertainty on the activity;
- Strong uncertainty on the timestamp;
- Strong uncertainty on indeterminate events.

All three can happen concurrently. Table III shows such a trace, which we will use as running example. It is worth noticing that the specific case of uncertainty on the case ID causes a problem; since an event can have many possible case IDs, it can belong to different traces. In data format where the event are already aggregated into traces, such as the very common XES standard, this means that the information related to a trace can be *non local* to the trace itself, but can be stored in some other points of the log. We will focus on the problem of uncertainty on the case ID attribute in a future work.

Firstly, we will lay down some simplified notation in order to model in a more compact way the problem at hand.

*Definition 8 (Simple traces and logs):* $\sigma_C \in \mathcal{U}_A^*$ is a *simple untimed trace*. $\mathcal{T}_C$ denotes the universe of simple untimed traces. $L_C^S \in \mathcal{B}(\mathcal{T}_C)$ is a *simple untimed log*.

$\sigma_{CT} \in (\mathcal{U}_A \times \mathcal{U}_T)^*$ is a *simple timed trace* if and only if $\sigma_{CT} = \langle (a_1, t_1), (a_2, t_2), \ldots, (a_n, t_n) \rangle$ and $\forall_{1 \leq i < j \leq n}, t_i < t_j$. $\mathcal{T}_{CT}$ denotes the universe of simple timed traces. $L_{CT}^S \in \mathcal{B}(\mathcal{T}_{CT})$ is a *simple timed log*.

$\sigma_U \in \mathcal{P}(\mathcal{U}_E \times \mathcal{P}_{NE}(\mathcal{U}_A) \times \mathcal{U}_T \times \mathcal{U}_T \times \mathcal{U}_O)$ is a *simple uncertain trace* if for all $(e, a_s, t_{min}, t_{max}, u) \in \sigma_U$, $t_{min} < t_{max}$ and all the event identifiers are unique. $\mathcal{T}_U$ denotes the universe of simple uncertain traces. $L_U^S \in \mathcal{B}(\mathcal{T}_U)$ is a *simple uncertain log* if all the event identifiers in the log are unique. For $e_U^S = (e, a_s, t_{min}, t_{max}, o) \in \sigma_U$ we define the following projection functions: $\pi_a^{L_U^S}(e_U^S) = a_s$, $\pi_{t_{min}}^{L_U^S}(e_U^S) = t_{min}$, $\pi_{t_{max}}^{L_U^S}(e_U^S) = t_{max}$ and $\pi_o^{L_U^S}(e_U^S) = o$.

*Definition 9 (Realization of a simple trace):* $\sigma_{CT} \in \mathcal{T}_{CT}$ is a *timed realization* of $\sigma_U \in \mathcal{T}_U$ if and only if:

- For all $(a, t) \in \sigma_{CT}$ there is a distinct $(a_s, t_{min}, t_{max}, u) \in \sigma_U$ such that $a \in a_s$ and $t_{min} \leq t \leq t_{max}$;
- For all $(a_s, t_{min}, t_{max}, u) \in \sigma_U$ with $u = !$ there is a distinct $(a, t) \in \sigma_{CT}$ such that $a \in a_s$ and $t_{min} \leq t \leq t_{max}$.

We denote with $\mathcal{R}_T(\sigma_U)$ the set of all such timed realizations of the trace $\sigma_U$. For $\sigma_{CT} \in \mathcal{T}_{CT}$ we denote with $\pi_A(\sigma_{CT})$ the simple untimed trace $\sigma_C \in \mathcal{T}_C$ such that $|\sigma_{CT}| = |\sigma_C| = n$ and for all $1 \leq i \leq n$, $\sigma_{CT}(i) = (a, t)$: $\sigma_C(i) = a$. For $\sigma_U \in \mathcal{T}_U$, $\mathcal{R}(\sigma_U) = \{\pi_A(\sigma_{CT}) \mid \sigma_{CT} \in \mathcal{R}_T(\sigma_U)\}$ is the set of all (untimed) *realizations* of $\sigma_U$.

These simplified traces and logs can be related to the more general framework described in the previous section through the following transformation: let $L_{SU} \subseteq \mathcal{E}_{SU}$ be a strongly uncertain log and let $g \colon \mathcal{U}_E \not\to \mathcal{U}_C$ be a function mapping events onto cases such that $dom(g) = \{e \mid (e, c_s, a_s, t_s, u) \in L_{SU}\}$ and for all $(e, c_s, a_s, t_s, u) \in L_{SU}$, $g(e) \in c_s$. Thus, for $c \in rng(g)$, $g^{-1}(c) = \{e \in \mathcal{U}_E \mid g(e) = c\}$. The simple uncertain event log defined by $g$ on $L_{SU}$ is $L_U^S = [\{(e, \pi_a^{\mathcal{E}_{SU}}(e), min(\pi_t^{\mathcal{E}_{SU}}(e)), max(\pi_t^{\mathcal{E}_{SU}}(e)), \pi_o^{\mathcal{E}_{SU}}(e)) \mid e \in g^{-1}(c)\} \mid c \in rng(g)]$.

The conformance checking algorithm that we are applying in this paper is based on *alignments*. Introduced by Adriansyah [11], conformance checking through alignments finds deviations between a trace and a Petri net model of a process by creating a correspondence between the sequence of activities executed in the trace and the firing of the transitions in the Petri net. An example of alignments is given in Figure 2.

*Definition 10 (System Net):* A system net is a tuple $SN = (P, T, F, l, M_{init}, M_{final})$ with $P$ the set of places, $T$ the set of transitions, $P \cap T = \emptyset$, $F \subseteq (P \times T) \cup (T \times P)$ the flow relation and $l \in T \not\to \mathcal{U}_A$ a labeling function over transitions. A marking $M \in \mathcal{B}(P)$ is a multiset of places; $M_{init} \in \mathcal{B}(P)$ is the initial marking of the net, and $M_{final} \in \mathcal{B}(P)$ is the final marking of the net. $\mathcal{U}_{SN}$ is the *universe of system nets*.

A system net $SN$ defines a directed graph with nodes $P \cup T$ and edges $F$. For any $x \in P \cup T$, $\bullet x = \{y \mid (y, x) \in F\}$ denotes the set of input nodes and $x \bullet = \{y \mid (x, y) \in F\}$ denotes the set of output nodes. A transition $t \in T$ is *enabled* in marking $M$ of net $SN$, denoted as $(SN, M)[t\rangle$, if each of its input places $\bullet t$ contains at least one token. An enabled transition $t$ may *fire*, i.e., one token is removed from each of the input places $\bullet t$ and one token is produced for each of the output places $t\bullet$. If $t \notin dom(l)$, it is called *invisible*. To indicate invisible transitions we use the placeholder symbol $\tau$; by definition $\tau \notin dom(l)$. An occurrence of visible transition $t \in dom(l)$ corresponds to observable activity $l(t)$. Given a system net, $\phi(SN)$ is the set of all possible *visible* activity sequences, i.e. the labels of complete firing sequences starting in $M_{init}$ and ending in $M_{final}$ projected onto the set of observable activities. Given the set of activity sequences $\phi(SN)$ obtainable via complete firing sequences on a certain system net, we can define a perfectly fitting event log as a set of traces which activity projection is contained in $\phi(SN)$.

These definitions allow us to build *alignments* in order to compute the fitness of trace on a certain model. An alignment is a correspondence between a sequence of activities (extracted from the trace) and a sequence of transitions with the relative labels (fired in the model while replaying the trace). The first sequence indicates the "moves in the log" and the second indicates the "moves in the model". If a move in the model cannot be mimicked by a move in the log, then a "$\gg$" ("no move") appears in the top row; conversely, if a move in the log cannot be mimicked by a move in the model, then a "$\gg$" ("no move") appears in the bottom row."no moves" not corresponding to invisible transitions point to deviations between model and log. A *move* is a pair $(x, (y, t))$ where the first element refers to the log and the second element to the model. A "$\gg$" in the first element of the pair indicates a move on model, in the second element it indicates a move on log.

An alignment is a sequence of moves such that after removing all "$\gg$" symbols, the top row corresponds to a trace in the log and the bottom row corresponds to a firing sequence starting in $M_{init}$ and ending $M_{final}$. Notice that if $t \notin dom(l)$ is an invisible transition, the activation of $t$ is indicated by a "$\gg$" on the log in correspondence of $t$ and the placeholder label $\tau$. Hence, the middle row corresponds to a visible path when ignoring the $\tau$ steps. Figure 2 shows a model with two examples of alignments, one of a fitting trace and the other of a non-fitting trace.

*Definition 11 (Alignment):* Let $\sigma_C \in L_C$ be a trace and $t \in \phi_f(SN)$ a complete firing sequence of system net $SN$. An *alignment* of $\sigma_C$ and $t_*$ is a sequence of moves $\gamma \in {A_M}^*$ such that the projection on the first element (ignoring $\gg$) yields $\sigma$ and the projection on the last element (ignoring $\gg$ and transition labels) yields $t$.

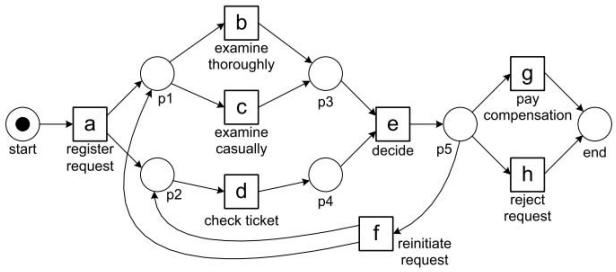
A trace and a model can have several possible alignments. In order to select the most appropriate one, we introduce a function that associate a *cost* to undesired moves - the ones associated with deviations.

*Definition 12 (Cost of Alignment):* Cost function $\delta \in A_{LM} \rightarrow \mathbb{N}$ assigns costs to legal moves. The *cost* of an alignment $\gamma \in {A_{LM}}^*$ is the sum of all costs: $\delta(\gamma) = \sum_{(x,y)\in\gamma} \delta(x, y)$.

In this paper we use a standard cost function $\delta_S$ that assigns cost zero to synchronous moves and moves on invisible transitions, and unit costs to moves on log or moves on model.

*Definition 13 (Optimal Alignment):* Let $L_C \in \mathcal{B}(\mathcal{T}_C)$ be a simple untimed event log and let $SN \in \mathcal{U}_{SN}$ be a system net with $\phi(SN) \neq \emptyset$.

- For $\sigma_C \in L_C$, we define: $\Gamma_{\sigma_C,SN} = \{\gamma \in {A_{LM}}^* \mid \exists_{t_* \in \phi_f(SN)}\ \gamma \textit{ is an alignment of } \sigma_C \textit{ and } t_*\}$.
- An alignment $\gamma \in \Gamma_{\sigma_C,SN}$ is *optimal* for trace $\sigma_C \in L_C$ and system net $SN$ if for any $\gamma' \in \Gamma_{\sigma_C,M}$: $\delta(\gamma') \geq \delta(\gamma)$.
- $\lambda_{SN} \in \mathcal{E}^* \rightarrow {A_{LM}}^*$ is a deterministic mapping that assigns any trace $\sigma_C$ to an optimal alignment, i.e., $\lambda_{SN}(\sigma_C) \in \Gamma_{\sigma_C,SN}$ and $\lambda_{SN}(\sigma_C)$ is optimal.
- $costs(L_C, SN, \delta) = \sum_{\sigma_C \in L} \delta(\lambda_{SN}(\sigma_C))$ are the *misalignment costs* of the whole event log.




$$\gamma_1 = \begin{array}{|c|c|c|c|c|}\hline a & d & b & e & h \\ \hline a & d & b & e & h \\ \hline \end{array} \qquad \gamma_2 = \begin{array}{|c|c|c|c|c|c|}\hline a & \gg & d & b & e & h \\ \hline a & b & d & b & e & h \\ \hline \end{array}$$

Fig. 2. Example of alignments on a model. The alignment $\gamma_1$ shows that the trace $\langle a, d, b, e, h\rangle$ is perfectly fitting the model. The alignment $\gamma_2$ shows that the trace $\langle a, b, d, b, e, h\rangle$ is misaligned with the model in one point.

TABLE III
THE UNCERTAIN TRACE USED AS RUNNING EXAMPLE FOR THE APPLICATION OF CONFORMANCE CHECKING ON UNCERTAINTY.

| Case ID | Timestamp | Activity | Indet. event |
|---|---|---|---|
| 0 | 2011-12-05T00:00 | A | ! |
| 0 | 2011-12-07T00:00 | {B, C} | ! |
| 0 | [2011-12-06T00:00 2011-12-10T00:00] | D | ! |
| 0 | 2011-12-09T00:00 | {A, C} | ! |
| 0 | 2011-12-11T00:00 | E | ? |

Depending on the possible values for $a_s$, $t_{min}$, $t_{max}$, and $u$ there are multiple possible realizations of a trace. This means that, given a model, a simple uncertain trace could be fitting for certain realizations, but non-fitting for others. The question we are interested in answering is: given a simple uncertain trace and a Petri net process model, is it possible to find an *upper and lower bound* for the conformance score? More formally, when usually we are interested in the optimal alignments (the ones with the minimal cost), we are now interested in the minimum and maximum cost of alignments in the realization set of a simple uncertain trace.

*Definition 14 (Upper and Lower Bound on Alignment Cost for a Trace):* Let $\sigma_U \in \mathcal{T}_U$ be a simple uncertain trace, and let $SN \in \mathcal{U}_{SN}$ be a system net. The *upper bound for the alignment cost* is a function $\delta_{max}\colon \mathcal{T}_U \rightarrow \mathbb{N}$ such that $\delta_{max}(\sigma_U) = \ max\ _{\sigma_C \in \mathcal{R}(\sigma_U)} \lambda_{SN}(\sigma_C)$. The *lower bound for the alignment cost* is a function $\delta_{min}\colon \mathcal{T}_U \rightarrow \mathbb{N}$ such that $\delta_{min}(\sigma_U) = \ min\ _{\sigma_C \in \mathcal{R}(\sigma_U)} \lambda_{SN}(\sigma_C)$.

A simple way to compute the upper and lower bounds for the cost of an uncertain trace is using a bruteforce approach: enumerating the possible realizations of the trace, then searching for the costs of optimal alignments for all the realizations, and picking the minimum and maximum as bounds.

The technique to compute the optimal alignment [11] is as follows. Firstly, it creates an *event net*, a sequence-structured system net able to replay only the trace to align. The transitions in the event net have labels corresponding to the activities in the trace. Then, a *product net* should be computed; it

is the union of the event net and the model together with synchronous transitions added. These additional transitions are paired with transitions in the event net and in the process model that have the same label; they are then connected with arcs from the input places and to the output places of those transitions. The product net is able to represent moves on log, moves on model and synchronous moves by means of firing transitions: the transitions of the event net correspond to moves on log, the transitions of the process model correspond to moves on model, the added synchronous transitions correspond to synchronous moves. The union of the initial and final markings of the event net and the process model constitute respectively the initial and final marking of the product net: every complete firing sequence on the product net corresponds to a possible alignment. Lastly, the product net is translated to a state space, and a state space exploration via the $\mathbb{A}^*$ algorithm is performed in order to find the complete firing sequence that yields the lowest cost.

Let us define formally the construction of the event net and the product net:

*Definition 15 (Event Net):* Let $\sigma_C \in \mathcal{T}_C$ be a simple untimed trace. The *event net* $en : \mathcal{T}_C \to \mathcal{U}_{SN}$ of $\sigma_C$ is a system net $en(\sigma_C) = (P, T, F, l, M_{init}, M_{final})$ such that:

- $P = \{p_i \mid 1 \leq i \leq |\sigma_C| + 1\}$,
- $T = \{t_i \mid 1 \leq i \leq |\sigma_C|\}$,
- $F = \bigcup_{1 \leq i \leq} \{(p_i, t_i), (t_i, p_{i+1})\}$
- $l \colon T \to \mathcal{U}_A$ such that for all $1 \leq i \leq |\sigma_C|$, $l(t_i) = \sigma_C[i]$,
- $M_{init} = \{p_1\}$,
- $M_{final} = \{p_{|P|}\}$.

*Definition 16 (Product of two Petri Nets):* Let $S_1 = (P_1, T_1, F_1, l_1, M_{init_1}, M_{final_1})$ and $S_2 = (P_2, T_2, F_2, l_2, M_{init_2}, M_{final_2})$ be two system nets. The *product net* of $S_1$ and $S_2$ is the system net $S = S_1 \otimes S_2 = (P, T, F, l, M_{init}, M_{final})$ such that:

- $P = P_1 \cup P_2$,
- $T \subseteq (T_1 \cup \{\gg\} \times T_2 \cup \{\gg\})$ such that $T = \{(t_1, \gg) \mid t_1 \in T_1\} \cup \{(\gg, t_2) \mid t_2 \in T_2\} \cup \{(t_1, t_2) \in (T_1 \times T_2) \mid l_1(t_1) = l_2(t_2) \neq \tau\}$,
- $F \subseteq (P \times T) \cup (T \times P)$ such that
  $F = \{(p, (t, \gg)) \mid p \in P_1 \wedge t \in T_1 \wedge (p, t) \in F_1\} \cup$
  $\{((t, \gg), p) \mid t \in T_1 \wedge p \in P_1 \wedge (t, p) \in F_1\} \cup$
  $\{(p, (t, \gg)) \mid p \in P_2 \wedge t \in T_2 \wedge (p, t) \in F_2\} \cup$
  $\{((t, \gg), p) \mid t \in T_2 \wedge p \in P_2 \wedge (t, p) \in F_2\} \cup$
  $\{(p, (t_1, t_2)) \mid p \in P_1 \cup P_2 \wedge (t_1, t_2) \in T \cap (T_1 \times T_2)\} \cup$
  $\{((t_1, t_2), p) \mid p \in P_1 \cup P_2 \wedge (t_1, t_2) \in T \cap (T_1 \times T_2)\}$
- $l \colon T \to \mathcal{U}_A$ such that for all $(t_1, t_2) \in T$, $l((t_1, t_2)) = l_1(t_1)$ if $t_2 = \gg$, $l((t1, t2)) = l_2(t_2)$ if $t_1 = \gg$, and $l((t_1, t_2)) = l_1(t_1)$ otherwise,
- $M_{init} = M_{init_1} \uplus M_{init_2}$,
- $M_{final} = M_{final_1} \uplus M_{final_2}$.

We now present a technique which improves the performance of calculating the lower bound for conformance cost over using a bruteforce method. We will produce a version of the event net that embeds the possible behaviors of the uncertain trace. We define a *behavior net*, a Petri net that can replay all and only the realizations of an uncertain trace.

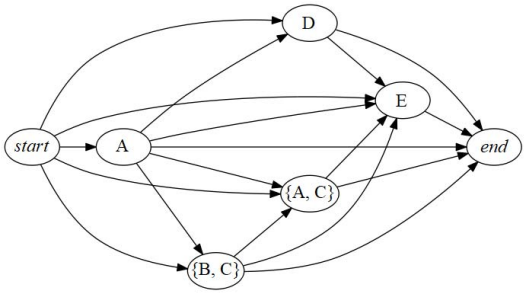



Fig. 3. The behavior graph of the trace in Table III before applying the transitive reduction.

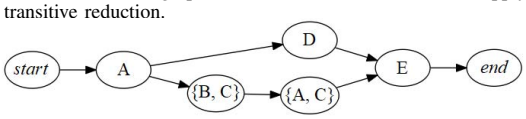



Fig. 4. The same behavior graph after the transitive reduction.

In order to obtain such a Petri net we first built a directed graph representing the uncertain trace as an intermediate step. We also need to present the concept of *transitive reduction*: given a directed graph $G$, its transitive reduction $G'$ is a graph with the same set of vertices, the same reachability between vertices, and a minimal number of arcs, such that every pair of vertices is connected by at most one path. The transitive reduction of a directed acyclic graph always exists and is unique [12].

We can then define the *behavior graph*, which contains a vertex for each uncertain event in the trace and contains an edge between two vertices if the corresponding uncertain events may happen one directly after the other.

*Definition 17 (Behavior Graph):* Let $\sigma_U \in \mathcal{T}_U$ be a simple uncertain trace. A *behavior graph* $bg \colon \mathcal{T}_U \to \mathcal{U}_G$ is the transitive reduction of a directed graph $(V, E)$, where $V$ is the set of vertices and $E \subseteq V \times V$ is the set of directed edges, such that:

- $V = \sigma_U$
- $E = \{(v_1, v_2) \mid v_1 \in V, v_2 \in V, \pi^{L_U^S}_{t_{max}}(v_1) < \pi^{L_U^S}_{t_{min}}(v_2)\}$

The behavior graph provides a structured representation of the uncertainty on the timestamp: when a specific vertex has two or more outbound edges, the events corresponding to the destination vertices can occur in any order, concurrently with each other. The property of $\mathcal{U}_T$ of being totally ordered and the property $t_{min} < t_{max}$ of all simple uncertain traces ensure that the behavior graph is acyclic both before and after the transitive reduction. We can see the result on the example trace in Figures 3 and 4.

We then obtain a *behavior net* by replacing every vertex in the behavior graph with one or more transitions in a XOR configuration, each representing an activity contained in the $\pi_{\mathcal{A}}$ set of the corresponding uncertain event. The edges of the behavior graph become connection through places in the behavior net.

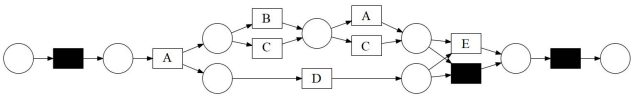



Fig. 5. The behavior net corresponding to the uncertain trace in Table III.

*Definition 18 (Behavior Net):* Let $\sigma_U \in \mathcal{T}_U$ be a simple uncertain trace, and let $\text{bg}(\sigma_U) = (V, E)$ be the corresponding behavior graph. A *behavior net* $bn\colon \mathcal{T}_U \to \mathcal{U}_{SN}$ is a system net $bn(\sigma_U) = (P, T, F, l, M_{init}, M_{final})$ such that:

- $T = \{(v, a) \mid v \in V \wedge a \in \pi_a^{L_U^S}(v)\} \cup \{(v, \tau) \mid v \in V \wedge a \in \pi_a^{L_U^S}(v) \wedge \pi_o^{L_U^S}(v) = ?\}$
- $P = E \cup \{\text{start}, \text{end}\}$
- $F = \{((v_1, a), (v_1, v_2)) \mid (v_1, a) \in T, (v_1, v_2) \in E\} \cup \{((v_1, v_2), (v_2, a)) \mid (v_1, v_2) \in P, (v_2, a) \in E\} \cup \{(\text{start}, (v, a)) \mid (v, a) \in T \wedge \forall_{v_* \in V} \pi_{t_{min}}^{L_U^S}(v) < \pi_{t_{min}}^{L_U^S}(v_*)\} \cup \{((v, a), \text{end}) \mid (v, a) \in T \wedge \forall_{v_* \in V} \pi_{t_{max}}^{L_U^S}(v) > \pi_{t_{max}}^{L_U^S}(v_*)\}$
- $l = \{((v, a), a) \mid (v, a) \in T \wedge a \neq \tau\}$
- $M_{\text{init}} = \{\text{start}\}$
- $M_{\text{final}} = \{\text{end}\}$

In Figure 5 we can see the behavior net corresponding to the uncertain trace in Table 2. It is important to notice that every set of edges in the behavior graph with the same source vertex generate an AND split in the behavior net, and a set of edges with the same destination vertex generate an AND join. At the same time, the transitions which labels correspond to different possible activities in an uncertain event will appear in a XOR construct inside the behavior net.

This means that every set of events which timestamps allow for overlapping will be represented in the behavior net by transitions inside an AND construct, and will then allow to execute in the net all the possible sequences of events obtained choosing a possible value for the uncertain timestamp attribute. In the same fashion, an event with uncertainty on the activity will be represented by a number of transitions in a XOR construct, that allows to replay any possible choice for the activity attribute. It follows that, by construction, for a certain simple uncertain trace $\sigma_U$ we have that $\phi(bn(\sigma_U)) = \mathcal{R}(\sigma_U)$.

We can use the behavior net of an uncertain trace $\sigma_U$ in lieu of the event net to compute alignments with a model $SN \in \mathcal{U}_{SN}$; the search algorithm returns an optimal alignment, a sequence of moves $(x, (y, t))$ with $x \in \mathcal{U}_A$, $y \in \mathcal{U}_A$ and $t$ transition of the model $SN$. After removing all "$\gg$" symbols, the sequence of first elements of the moves will describe a complete firing sequence $\sigma_X$ of the behavior net. Since $\sigma_X$ is complete, $\sigma_X \in \phi(bn(\sigma_U))$ and, thus, $\sigma_X \in \mathcal{R}(\sigma_U)$. It follows that $\sigma_X$ is a realization of $\sigma_U$, and the search algorithm ensures that $\sigma_X$ is a realization with optimal conformance cost for the model $SN$: $\delta(\lambda_{SN}(\sigma_X)) = \min_{\sigma_C \in \mathcal{R}(\sigma_U)} \lambda_{SN}(\sigma_C) = \delta_{min}(\sigma_U)$.

## VI. Experiments

The technique to compute conformance for strongly uncertain traces and to create the behavior net hereby described has been implemented for testing, using the code for alignments already provided in the process mining Python library PM4Py [13]. Uncertainty has been represented in the XES standard through meta-attributes and constructs such as lists, such that any XES importer can read an uncertain log file. The algorithm was designed to be fully compatible with non-uncertain XES event logs; the meta-attributes for uncertainty were designed to be partly compatible with other process mining algorithms – meta-attributes describing the possible values for an uncertain activity or the interval of an uncertain timestamp can also specify a "fallback value" that other process mining software will read as (certain) activity or timestamp value.

Two experiments were run: the first to inspect the bounds for conformance score as increasingly more uncertainty is added to an event log; the other test assesses the difference in performance between the bruteforce method and the behavior net. We ran the tests on synthetic event logs where we added uncertainty. This way we can control the amounts of uncertainty in event data. Through the ProM plugin *"Generate block-structured stochastic Petri nets"* we generated Petri nets of different sizes in terms of number of transitions $n$; then, we used PM4Py in order to generate event logs adding uncertainty to attributes. Activities and timestamps are uncertain with probability $p$; also, events have a chance to be indeterminate with probability $p$.

The pipeline for the first experiment was the following: we generated a model with $n$ = 10; we executed the model to obtain 250 traces; we added deviations to events (every event has 20% chances to have the wrong activity, every pair of consecutive events have 20% chances of having the timestamps swapped, every trace has 40% chances to have an additional event). We then added uncertainty to the events: each event has probability $p$ of having two possible values for the activity, probability $p$ of having an uncertain timestamp, and probability $p$ to be an indeterminate event. We calculated the bounds on conformance cost of the log so generated and repeated the procedure for increasing values of $p$.

Figure 6 shows the results. We can see that the cost shows the expected behavior: at $p$ = 0 the two bounds coincide, since the traces are certain and have only one realization. Conversely, the log with $p$ = 0.6 has a total of 1629 deviations on the worst case scenario (6.52 on average per trace), and 747 deviations in the best case scenario (2.99 on average per trace); a process that includes traces with comparable uncertainty has thus a huge difference in behavior between the best case scenario and worst case scenario. The evaluation of the best and worst case scenario for uncertain traces can give useful indications to a business user on the parts of the process where there is a high risk of deviation, in order to enhance them.

The second experiment concerns the performance of calculating the lower bound of the cost via the behavior net

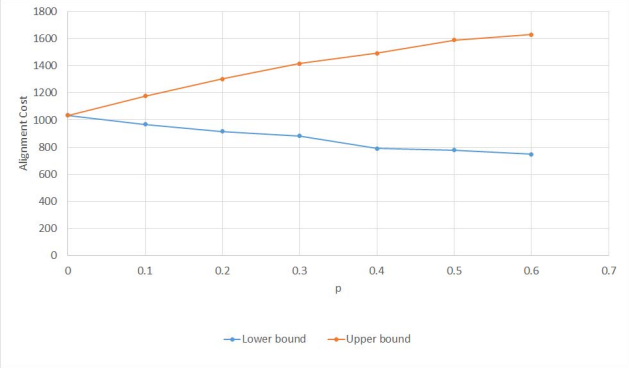



Fig. 6. The change in lower and upper bound for conformance checking of an event log with increasing probability of having uncertainty on event data.

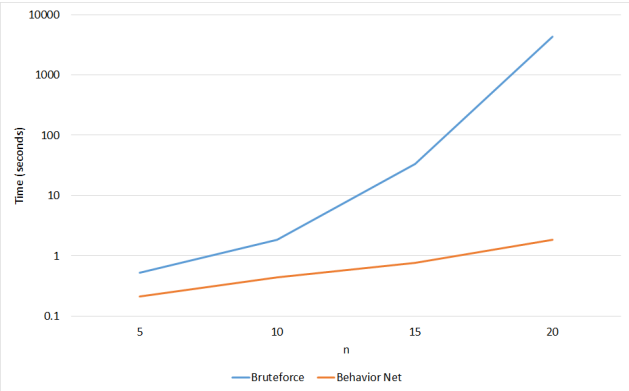



Fig. 7. Effect on time performance of calculating the lower bound for conformance cost with the bruteforce method vs. the behavior net.

versus the bruteforce method of separately listing all the realizations of an uncertain trace, evaluating all of them through alignments, then picking the best value. We used a constant value of $p = 0.2$ and logs of 100 traces for this test, with progressively increasing values of $n$.

Figure 7 summarizes the results. As the diagram shows, the difference in time between the two methods tends to diverge quickly even on a logarithmic scale. With $n$ = 5, the behavior net provides the lower bound in 40.4% of the time required by the bruteforce method. For $n$ = 20, the largest model we could test, the behavior net takes 0.04% of the time needed by the bruteforce method. This shows a very large improvement in the computing time for the lower bound computation, so the best case scenario for the conformance cost of an uncertain trace can be obtained efficiently thanks to the structural properties of the behavior net.

## VII. Conclusion

As the need of quickly and effectively analyze process data has arisen in the recent past and is growing to this day, many new types of information regarding events are recorded; this calls for new techniques able to provide an adequate interpretation of the new data. In this paper we presented a new paradigm for process mining applied to event data: explicit uncertainty. We described the possible form it can assume, building a taxonomy of different types of uncertainty. We then designed a formal mathematical infrastructure to define the various flavors of uncertainty shown in the taxonomy. Then, in order to assess the practical applications of the uncertainty framework, we applied it to a well consolidated technique for conformance checking: aligning data to a reference Petri net. The results can provide insights on the possible violations of process instances recorded with uncertainty against a normative model. The behavior net provides an efficient way to compute the lower bound for the conformance cost – i.e. the best case scenario for conformity of uncertain process data – with a large improvement on time performance with respect to a bruteforce procedure.

The approaches shown here can be extended in a number of ways. An important step in this line of research is assessing the technique on real-life logs. From a performance perspective, to improve the usability of alignments over uncertainty we shall optimize the computation of the upper bound of the conformance cost. Another natural continuation of this work is extending the conformance checking technique to logs with weak uncertainty. Many possibilities can be pursued to broaden the concept of uncertainty on different process mining methods: for example, discovering a Petri net from an uncertain event log, or develop techniques to mine Petri nets that embed uncertainty information about the process.

## References

[1] W. Van der Aalst, A. Adriansyah, A. K. A. De Medeiros, F. Arcieri, T. Baier, T. Blickle, J. C. Bose, P. Van Den Brand, R. Brandtjen, J. Buijs *et al.*, "Process mining manifesto," in *International Conference on Business Process Management*. Springer, 2011, pp. 169–194.

[2] J. Carmona, B. van Dongen, A. Solti, and M. Weidlich, *Conformance Checking: Relating Processes and Models*. Springer, 2018.

[3] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.

[4] S. Suriadi, R. Andrews, A. H. ter Hofstede, and M. T. Wynn, "Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs," *Information Systems*, vol. 64, pp. 132–150, 2017.

[5] J. Wang, S. Song, X. Lin, X. Zhu, and J. Pei, "Cleaning structured event logs: A graph repair approach," in *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*. IEEE, 2015, pp. 30–41.

[6] R. Conforti, M. La Rosa, and A. H. ter Hofstede, "Filtering out infrequent behavior from business process event logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 2, pp. 300–314, 2017.

[7] M. F. Sani, S. J. van Zelst, and W. M. van der Aalst, "Improving process discovery results by filtering outliers using conditional behavioural probabilities," in *International Conference on Business Process Management*. Springer, 2017, pp. 216–229.

[8] S. J. van Zelst, M. F. Sani, A. Ostovar, R. Conforti, and M. La Rosa, "Filtering spurious events from event streams of business processes," in *International Conference on Advanced Information Systems Engineering*. Springer, 2018, pp. 35–52.

[9] R. Conforti, M. La Rosa, and A. ter Hofstede, "Timestamp repair for business process event logs," 2018, [preprint]. [Online]. Available: http://hdl.handle.net/11343/209011

[10] M. F. Sani, S. J. van Zelst, and W. M. van der Aalst, "Repairing outlier behaviour in event logs," in *International Conference on Business Information Systems*. Springer, 2018, pp. 115–131.

[11] A. Adriansyah, "Aligning observed and modeled behavior," 2014.

[12] A. V. Aho, M. R. Garey, and J. D. Ullman, "The transitive reduction of a directed graph," *SIAM Journal on Computing*, vol. 1, no. 2, pp. 131–137, 1972.

[13] "Pm4py." [Online]. Available: http://pm4py.pads.rwth-aachen.de/

# Measuring the Behavioral Quality of Log Sampling

Bram Knols and Jan Martijn E. M. van der Werf
Department of Information and Computing Sciences
Utrecht University
Princetonplein 5, 3584 CC Utrecht, The Netherlands
{b.knols, j.m.e.m.vanderwerf}@uu.nl

***Abstract*—Process mining combines data mining with process analysis, e.g. to discover process models from event logs. Practice shows that event logs grow very fast. Consequently, they quickly become too large to analyze with current tools. Given the exploratory nature of many process mining algorithms, this can be problematic, as in many cases algorithms are used frequently to optimize and analyze the influence of parameters. One solution is reducing the data by sampling the event log. Many sampling approaches exist, yet the quality of these approaches is unknown.**

**In this paper, we study the behavioral quality of event log sampling, and introduce measures to quantify this behavioral quality. The approach has been implemented in the tool ProM. Experiments show that sampling very quickly introduces under and oversampled behavior in the event log, which can be problematic for frequency-based algorithms.**

***Index Terms*—Event logs, Sampling, Behavioral quality, Quality measures**

## I. Introduction

Process Mining is a discipline at the intersection of data analysis and business process management [1]. Many different algorithms have been proposed to discover (cf. [2]–[7]), analyze (cf. [8]–[11]), and enrich business processes (cf. [12], [13]) based on event logs. An event log captures the execution of systems, containing information about e.g. which activities have been performed, when and by whom [14].

As systems keep running, event logs become rapidly very large. Event logs can be considered to be samples of an infinite stream of events generated by the system, as depicted in Figure 1. Each event log only registers a part of the event stream. For example, creating weekly event logs in an ERP system, in one week events of generating the payslips for all employees may be overrepresented, whereas in other weeks this activity rarely occurs. As such, event logs may introduce bias caused by the time window. However, as event logs are continuously augmented, sampling is inevitable. For example, Langerak et al. [15] studied a large parcel distributor. Although the process itself is relatively small, with on average 10 events per parcel, the distributor handled over 400M parcels a year. Due to the computational complexity of the algorithms, samples had to be taken for analysis. As this study exemplifies, the efficiency and complexity of model discovery and analysis is becoming a bottleneck [16].

Several methods have been proposed to attack the efficiency of process mining. For example, one can improve the efficiency of algorithms. However, as most algorithms scale at least linearly in size of the event log, this is only possible to a certain extent. Another approach is to reduce the event data to be analyzed, e.g., by sampling the event log. Sampling has the potential to solve the efficiency problems and allow existing algorithms to perform within acceptable time boundaries. However, the quality of the results rely on the quality of the sampling technique applied.

While the need for sampling event logs in research is acknowledged, no definite conclusions have been drawn. Most research is focused on either the filtering of event data or sampling event data for specific algorithms. Both of these approaches are useful, but can not be generalized beyond their scope and present little useful information on the quality of the samples themselves.

Sampling has been used to combat the so called state-explosion problem in Process Mining (cf. [17], [18]). However, in these studies the effect of sampling itself is not studied, merely it is applied to improve the efficiency of algorithms.

Other techniques, such as proposed in [11], [19] remove outliers with filtering, thereby increasing the quality of the models created. However useful for improving model quality, these techniques does not significantly reduce the amount of event data, and thus the size of the sample.

Berti [20] studies sampling with the Heuristics Miner [7]. He concludes that about 3% of the log is enough to extract over 95% of the dependencies. While relevant, the aim of our research is to provide more insight on the behavioral quality of event log sampling.

The statistical framework introduced by [21], also based on the directly-follows relation, samples the event log based on how much new information each trace adds. While their approach improves efficiency and is stated to work for a range

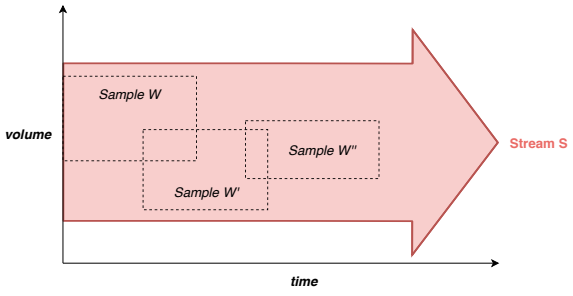



Fig. 1. Event logs seen as samples of an infinite stream of events

978-1-7281-0919-0/19/$31.00 ©2019 IEEE
DOI 10.1109/ICPM.2019.00024

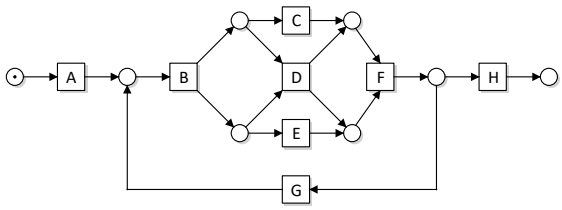



Fig. 2. Example Petri Net model

TABLE I
THE EXAMPLE EVENT LOG

| Week | Seq. | Trace | Freq. |
|---|---|---|---|
| 1 | 1 | A, B, C, E, F, H | 3 |
| | 2 | A, B, C, E, F, G, B, E, C, F, H | 4 |
| 2 | 3 | A, B, D, F, H | 9 |
| 3 | 4 | A, B, C, E, F, H | 3 |
| | 5 | A, B, C, E, F, G, B, E, C, F, H | 2 |

of algorithms, they do not study the sample quality. Instead, their approach is somewhere between sampling and filtering, as they create a sample that satisfies a given constraint. Another approach based on the directly-follows relation is taken in [22]. The authors study the statistical properties of traces generated by different model classes. And, assuming that a specific model class generated an event log, provide a measure on the completeness of that event log. The authors of [23] study global completeness of event logs. Their measures are on the level of traces. For example, they propose the Observed Trace variants Ratio (OTR) as the ratio between the number of trace variants observed and the total number of trace variants generated by the process, and augment it with the accumulated coverage probability (ACP), i.e., the chance that a next trace is already part of the event log.

Leemans et al. [24] mentions log sampling to reduce the memory size, but states that this may result in over or under fitting a model because of the incomplete logs. While apparently a problem, this poses as a strong motivation for the research we present in this paper.

In this paper, we study the behavioral quality of sampling event logs. Quality measures aid in providing insights in the potential information loss of samples. For process mining, such measures should focus on quantifying quality of the sampled behavior. We propose six measures that, based on the original event log, quantifies which behavior is sampled optimally, and whether behavior is under or oversampled.

The remainder of this paper is structured as follows. First, Section II introduces the basic notions for process mining used in this paper. Section III presents the ideas and techniques we propose to measure the behavioral quality of log samples. To validate our approach, we report on initial experiments on two BPI Challenge logs, which are presented in Section IV, and discussed in Section V. Last, Section VI wraps up the paper and presents possibilities for future work.

## II. Basic Notions

Let $S$ be a (possibly infinite) set. A *bag* or *multiset* over $S$ is a function $m : S \to \mathbb{N}$, where $\mathbb{N} = \{0, 1, 2, \ldots\}$ denotes the set of natural numbers. For $s \in S$, $m(s)$ denotes the number of occurrences of $s$ in $m$. The length of bag $m$, denoted by $\|m\|$, is defined as the sum of the count of its elements, i.e., $\|m\| = \sum_{s \in S} m(s)$. We use $\emptyset$ to denote the empty bag, and $\in$ to denote the element inclusion operation over bags. The set of all bags over $S$ is denoted by $\mathbb{B}(S)$.

A *sequence* over $S$ of length $n \in \mathbb{N}$ is a function $\sigma : \{1, \ldots, n\} \to S$. If $n > 0$ and $\sigma(i) = a_i$ for $i \in \{1, \ldots, n\}$, we write $\sigma = \langle a_1, \ldots, a_n \rangle$. The length of a sequence $\sigma$ is denoted by $\|\sigma\|$. The sequence of length 0 is called the *empty sequence*, and is denoted by $\epsilon$. The set of all finite sequences over $S$ is denoted by $S^*$. We write $a \in \sigma$ if some $1 \leq i \leq \|\sigma\|$ exists such that $\sigma(i) = a$.

**Definition 1** (Event log, trace)**.** *Given a set of activities $A$, an* Event Log *$L$ is defined as a bag over finite sequences over $A$, i.e., $L \subseteq \mathbb{B}(A^*)$. An element $\sigma \in L$ is called a* trace*.*

The directly-follows relation expresses the direct successorship of activity occurrences in an event log, i.e., given an event log $L$, activity $b \in A$ directly follows activity $a \in A$, denoted by $a <_L b$ iff a sequence $\sigma \in L$ and $1 \leq i < \|\sigma\|$ exist such that $\sigma(i) = a$ and $\sigma(i + 1) = b$. If the context is clear, we omit the subscript.

**Definition 2** (Behavior)**.** *Given an event log $L$, its* behavior*, denoted by $\mathcal{B}(L)$, is defined as all pairs of activities that occur consecutively, i.e., $\mathcal{B}(L) = \{(x, y) \mid x <_L y\}$. An element $(a, b) \in \mathcal{B}(L)$ is called a* behavior*.*

Given some sequence $\sigma \in L$, the frequency of some behavior $(a, b)$ is defined by the count of that behavior in $\sigma$, i.e., $f_\sigma(a, b) = \|\{0 \leq i < \|\sigma\| \mid \sigma(i) = a \wedge \sigma(i+1) = b\}\|$. We lift frequencies to event logs. Given an event log $L$, we define $f_L(a, b) = \sum_{\sigma \in L} L(\sigma) \cdot f_\sigma(a, b)$.

## III. Measures for Log Sampling

This section explores event log sampling and proposes six measures to evaluate the behavioral quality of a sample. We use an example to illustrate problems with sampling, and how the proposed measures can help identifying these problems.

### A. Sampling Event Logs

In our approach, we consider a sample of an event log to be a subset of the traces observed in the original log. A sample log contains each trace at most as often as observed in the original. Ideally, this sample has the same characteristics as the original event log.

**Definition 3** (Sample log)**.** *Let $L \in \mathbb{B}(A)$ be some event log over the set of activities $A$. An event log $S$ is a* sample log *of $L$ iff $S(\sigma) \leq L(\sigma)$, for all traces $\sigma \in L$.*

Consequently, the behavior of a sample is at most the behavior of an event log. To illustrate potential pitfalls of log sampling,

we use an example. Consider the Petri net model of Figure 2. In a standard week, activities $C$ and $E$ are executed in parallel after activity $B$, and either this process is repeated (activity $G$), or the process is terminated (activity $H$). In a busy period, activities $C$ and $E$ are replaced by a single activity $D$, and the process is never repeated. Monitoring the system for a large interval, the system first logs a quiet period, followed by a busy period, and concluded with another quiet period, as shown in the event log in Table I. Discovering a process model using the $\alpha$-Miner [3] results in the initial process model depicted in Figure 2. This Petri net clearly displays all behavior shown in the event log.

Now, suppose we sample the event log for the different periods, i.e., sample event log $S_1$ represents the first quiet period containing traces 1 and 2. Sample event log $S_2$ represents the busy period having only trace 3, and sample event log $S_3$ recorded the second quiet period with traces 4 and 5. One easily sees that these samples do not represent the complete log well. Sample log $S_2$ includes only path $D$ through the model, while samples $S_1$ and $S_2$ contain activities $C$ and $E$ but not activity $D$. In addition, the loop caused by activity $G$ in the model occurs only in the first and last sample. The behavior of these samples is therefore clearly different from the original, complete, event log. As the above example shows, samples may introduce bias to certain observed behavior. For example, in $S_2$, the behavior $(B, D)$ is oversampled with respect to the complete log.

To quantify the difference in behavior, a first measure would be the fraction of behavior in both the sample and original log:

$$F_L(S) = \frac{\|\mathcal{B}(S)\|}{\|\mathcal{B}(L)\|} \tag{1}$$

In our example, the original event log of Table I has 11 behaviors, sample $S_1$ and $S_2$ have both 9 behaviors, and sample $S_2$ has only 4 behaviors. This results in $F_L(S_1) = F_L(S_3) = \frac{9}{11} \approx 0,81$ and $F_L(S_2) = \frac{4}{11} \approx 0.36$. This measure hints that samples $S_1$ and $S_2$ are better samples, as the measure is closer to 1. However, this measure does not take frequencies into account. Samples $S_1$ and $S_3$ are valued equally, whereas their frequencies differ. In the remainder of this section, we study how frequencies can be used to define measures to quantify the behavioral quality of sample logs.

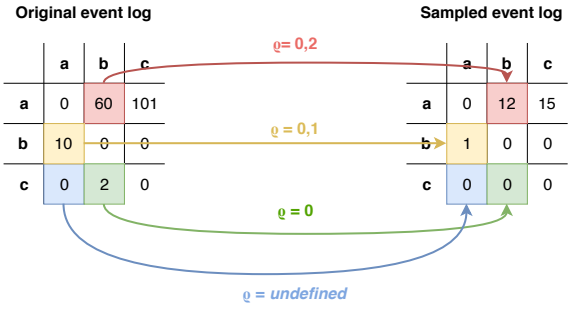



Fig. 3. Illustration of the sample ratio matrix construction

### B. Behavior-Based Sample Ratio

Many process mining algorithms are based on the directly-follows relation. Coarsely, two classes of algorithms can be distinguished: *plain algorithms* that are based on the existence of an element in the directly-follows relation, and *frequency-based algorithms* that take the number of occurrences of these elements into account. Examples of the former class include the $\alpha$-Miner [3], the ILP Miner [4] and the Inductive Miner [5]. Examples of the latter class are the Genetic [6] and Heuristic [7] Miner, but also filtering techniques that are used as preprocessing steps [25], [26], such as noise [19] and outlier detection [11].

When studying the behavioral quality of event logs for plain algorithms, only the existence of behavior needs to be considered. Consequently, Equation 1 provides a sufficient measure, as the samples $S_1$ and $S_3$ exemplify. For frequency-based algorithms, a more precise measure is required that takes the occurrences of behavior into account.

As a first step, we calculate the frequency-matrix of the directly-follows relation for the original event log. This results in a table, as shown in Table II for the original event log, and for samples $S_1$ and $S_2$ in Table III. In a sample log, each behavior occurs at most as often as in the original log, since otherwise it is not a sample according to Definition 3. Hence, we can calculate the relative occurrence of each behavior in relation to the original log. The general idea of this approach is depicted in Figure 3. This approach results in the sample ratio per behavior.

**Definition 4** (Sample Ratio for a Behavior)**.** *Given sample log $S$ of event log $L$, the* sample ratio*, denoted by $\rho_L^S$, of behavior $(a, b) \in \mathcal{B}(L)$ is defined as:*

$$\rho_L^S(a, b) = \frac{f_S(a, b)}{f_L(a, b)} \tag{2}$$

In case the context is clear, the subscript and superscript are omitted. Given a sample $S$ of some event log $L$, each behavior in the event log has a sample ratio between 0 and 1, as $S$ is a sample of $L$. Note that undefined behavior in the original event log is not taken into account in this ratio, and hence no undefined sample ratios will occur, since always $f_L(a, b) > 0$ for any $(a, b) \in \mathcal{B}(L)$.

In our example, behavior $(B, D)$ does not occur in sample $S_1$, which is visible in the sample ratio: $\rho_L^{S_1}(B, D) = \frac{0}{9} = 0$. Similarly, from sample rate $\rho_L^{S_2}(B, D) = \frac{9}{9} = 1$, we can conclude that all $(B, D)$ behavior is only present in sample $S_2$. Other behavior, such as $(A, B)$ has better ratios in both samples: $\rho_L^{S_1}(A, B) = \frac{7}{21} \approx 0.33$ and $\rho_L^{S_2}(A, B) = \frac{9}{21} \approx 0.43$. The complete matrix of all sample ratios for samples $S_1$ and $S_2$ are given in Tables IV and V.

The sample ratio matrix provides for each behavior the ratio between the original and the sample log. In itself, this value is non-descriptive, as it a relative value that varies over the different behaviors. We propose to match it with the *expected ratio* $r$, defined by the traces contained in the sample compared to the event log:

TABLE II
THE DIRECTLY-FOLLOWS FREQUENCY MATRIX FOR THE ORIGINAL LOG

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| A | | 21 | | | | | | |
| B | | | 12 | 9 | 6 | | | |
| C | | | | | 12 | 6 | | |
| D | | | | | | 9 | | |
| E | | | | | | 12 | | |
| F | | | | | | | 6 | 21 |
| G | | 6 | | | | | | |

TABLE III
THE DIRECTLY-FOLLOWS FREQUENCY MATRIX OF SAMPLES $S_1$ AND $S_2$

$S_1$

| | B | C | E | F | G | H |
|---|---|---|---|---|---|---|
| **A** | 7 | | | | | |
| **B** | | 7 | 4 | | | |
| **C** | | | 7 | 4 | | |
| **D** | | | | | | |
| **E** | | 4 | | | | |
| **F** | | | | | 4 | 7 |
| **G** | 4 | | | | | |

$S_2$

| B | D | F | H |
|---|---|---|---|
| 9 | | | |
| | 9 | | |
| | | | |
| | | 9 | |
| | | | |
| | | | 9 |
| | | | |

TABLE IV
SAMPLE RATIO MATRIX FOR SAMPLE $S_1$

| | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|
| **A** | 0.33 | | | | | | |
| **B** | | 0.58 | 0 | 0.67 | | | |
| **C** | | | | 0.58 | 0.67 | | |
| **D** | | | | | 0 | | |
| **E** | | | | | 0.58 | | |
| **F** | 0.67 | | | | | 0.67 | 0.33 |
| **G** | | | | | | | |

TABLE V
SAMPLE RATIO MATRIX FOR SAMPLE $S_2$

| | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|
| **A** | 0.43 | | | | | | |
| **B** | | 0 | 1 | 0 | | | |
| **C** | | | | 0 | 0 | | |
| **D** | | | | | 1 | | |
| **E** | | | | | 0 | | |
| **F** | 0 | | | | | 0 | 0.43 |
| **G** | | | | | | | |

$$r_L^S = \frac{\|S\|}{\|L\|} \tag{3}$$

In case the context is clear, the subscript and superscript are omitted. In our example, sample $S_1$ has an expected ratio of $r_L^{S_1} = \frac{7}{21} \approx 0.33$, and sample $S_2$ has expected ratio $r_L^{S_2} = \frac{9}{21} \approx 0.43$. With these expected values, the sample ratio matrices of Tables IV and V provide more information: the closer a value is to the expected sample ratio, the more optimal the sample is. In this case, behavior $(A, B)$ is sampled optimally in both samples. Behavior $(B, D)$ is however undersampled in $S_1$ and oversampled in $S_2$.

The sample ratio provides a statistical distribution of values [27], which can be plotted. A theoretical plot is shown in Figure 4, in which all behavior is decreasingly ordered based on its ratio. Notice that we plot all behavior, including undefined behavior. The expected ratio is visualized as well. A sample is sampled optimally if all ratios are equal to the expected value. However, in reality we suspect this rarely occurs. If for some behavior, its ratio is sufficiently close to the expected ratio, the behavior is said to be *truly sampled*. The accepted closeness is defined as a small interval around the expected value, the *true sample-bandwidth*. In this way, we construct a classification per behavior:

- *Non-existing behavior ($\mathcal{C}$)* not seen in the original event log $L$, and hence no ratio exists (i.e., all pairs that are not in $\mathcal{B}(L)$);
- *Oversampled behavior ($\mathcal{O}$)* with a ratio larger than the true sample-bandwidth;
- *Truly sampled behavior ($\mathcal{T}$)* with a ratio within the true sample-bandwidth; and
- *Undersampled behavior ($\mathcal{U}$)* with a ratio smaller than the true sample-bandwidth.

To better understand the undersampled behavior, we define in addition the set of *unsampled behavior ($\mathcal{N}$)* as all behavior unseen in the sample, i.e., with sample ratio zero. Note that this set is a subset of the undersampled behavior.

### C. Behavioral Quality Measures for Samples

Based on the sample ratio and theoretical plot, we propose six different measures to quantify the behavioral quality of a sample:

1) the mean sample ratio $\mu$;
2) the standard deviation $\sigma$ of the sample ratio;
3) the percentage of unsampled behavior ($P_{\mathcal{N}}$);
4) the percentage of undersampled behavior ($P_{\mathcal{U}}$);
5) the percentage of oversampled behavior ($P_{\mathcal{O}}$); and
6) the percentage of truly sampled behavior ($P_{\mathcal{T}}$).

The first two measures, the sample ratio mean and its standard deviation, are default statistical metrics, providing the first two moments of the distribution formed by the sample ratios [27]. In addition, the sample ratio mean gives a general indication of under and oversampling of the sample: if $\mu > r$, or $\mu < r$, then the complete log is oversampled, or undersampled, respectively. The standard deviation provides a general measure of variance in the sample.

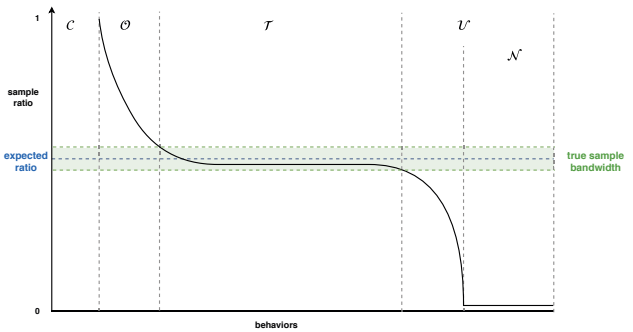



Fig. 4. Theorized plot of the sorted sample ratio matrix

TABLE VI
COUNT OF THE DIFFERENT CLASSES OF SAMPLE LOG $S_1$ AND $S_2$

| **Class** | **Sample 1** | **Sample 2** |
|---|---|---|
| Behavior ($\mathcal{B}$) | 9 | 4 |
| Oversampled ($\mathcal{O}$) | 2 | 2 |
| Truly sampled ($\mathcal{T}$) | 2 | 2 |
| Undersampled ($\mathcal{U}$) | 7 | 7 |
| Unsampled ($\mathcal{N}$) | 2 | 7 |

TABLE VII
QUALITY MEASURES OF SAMPLE LOGS $S_1$ AND $S_2$

| **Measure** | **Sample 1** | **Sample 2** |
|---|---|---|
| $r$ | 0.33 | 0.43 |
| $\mu$ | 0.46 | 0.26 |
| $\sigma$ | 0.26 | 0.40 |
| truly sampled ($P_{\mathcal{T}}$) | 18% | 18% |
| Oversampled ($P_{\mathcal{O}}$) | 18% | 18% |
| Undersampled ($P_{\mathcal{U}}$) | 82% | 64% |
| Unsampled ($P_{\mathcal{N}}$) | 18% | 64% |

The other measures are defined as a characteristic divided by the total observed behavior. The unsampled behavior $P_{\mathcal{N}}$ (Equation 4) describes how much behavior is observed in the original log, but is completely absent in the sample.

$$P_{\mathcal{N}} = \frac{\|\mathcal{N}\|}{\|\mathcal{B}(L)\|} \times 100\% \tag{4}$$

The undersampled behavior $P_{\mathcal{U}}$ (Equation 5) defines the percentage of undersampled behavior seen in the sample. Note that this includes the unsampled behavior.

$$P_{\mathcal{U}} = \frac{\|\mathcal{U}\|}{\|\mathcal{B}(L)\|} \times 100\% \tag{5}$$

Similarly, $P_{\mathcal{O}}$ (Equation 6) defines the percentage of oversampled behavior seen in the sample:

$$P_{\mathcal{O}} = \frac{\|\mathcal{O}\|}{\|\mathcal{B}(L)\|} \times 100\% \tag{6}$$

The last measure is the truly sampled behavior, $P_{\mathcal{T}}$, given in Equation 7. The larger this percentage is, the more behavior is truly sampled.

$$P_{\mathcal{T}} = \frac{\|\mathcal{T}\|}{\|\mathcal{B}(L)\|} \times 100\% \tag{7}$$

The general idea behind these measures is that a high $P_{\mathcal{T}}$ in combination with $\mu$ and $r$ being close together indicate that the sample can be used in process mining to make statements about the complete process. The other measures provide an overall impression of the behavioral quality of the sample.

To illustrate these measures, consider again the example. Table VI shows the behavior for each class, and table VII shows the quality measures derived from these values. We used a true sample bandwidth of $0.2$ in this example (i.e., the band has interval $r \pm 0.1$). In sample $S_2$, 64% of all behavior is unsampled, as it only contained a single, short trace. Consequently, as all measures indicate, it is not a true sample. First of all, the mean sample ratio is smaller than the expected ratio $r$, and secondly, the truly sampled

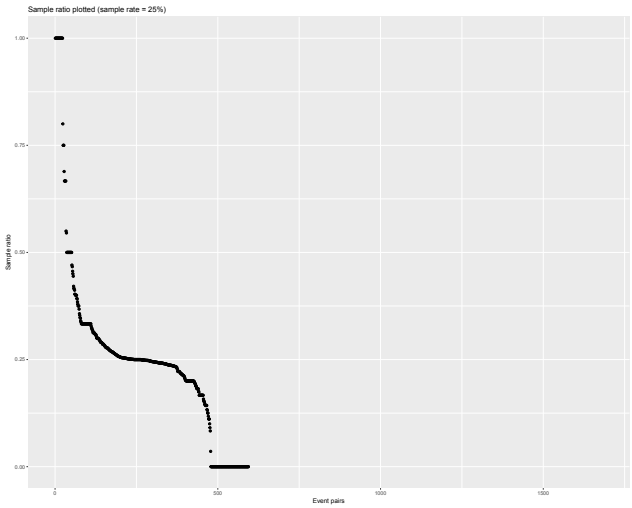



Fig. 5. Plot of a sample with a sample rate of 0.25. Note that it follows the theorized plot in Figure 4.

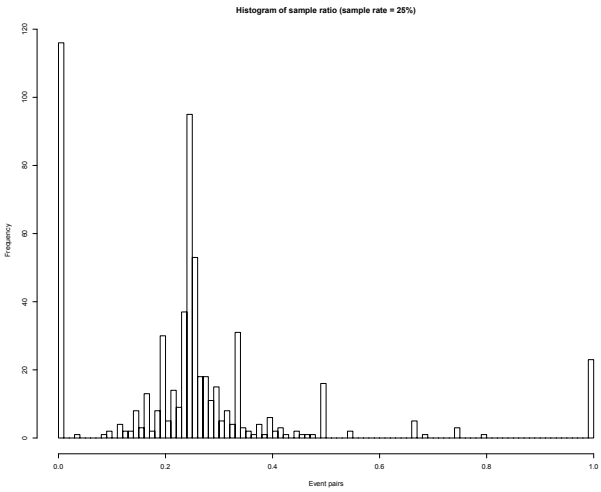



Fig. 6. Distribution of the sample ratio $\rho$

percentage is very low with $P_{\mathcal{T}}(S_2) = 0.18$. Similarly, sample $S_1$ is oversampled, as the mean sample ratio is larger than the expected ratio $r$. Both samples contain large sets of unsampled behavior. The measures indicate that both samples are inappropriate to analyze the complete process.

## IV. INITIAL EXPERIMENTS

To evaluate the proposed quality measures, we set up two experiments. In each experiment, we sample an event log with several values for sample rate $r$ and compute the quality measures. We use the process mining tool ProM [14] for the implementation, as it allows users to build their own plugins. Furthermore, it has several built-in features that facilitate the processing and analysis of event logs. This section explains the setup of the experiment and reports the initial results.

TABLE VIII
QUALITY MEASURES FOR DIFFERENT $r$ VALUES ($k = 10$) OF THE BPI CHALLENGE 2018 LOG

| r | **Mean ratio** ($\mu$) | | **Std. Dev.** ($\sigma$) | | **Truly sampled** ($P_{\mathcal{T}}$) | | **Oversampled** ($P_{\mathcal{O}}$) | | **Undersampled** ($P_{\mathcal{U}}$) | | **Unsampled** ($P_{\mathcal{N}}$) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | mean | std. dev. | mean | std. dev. | mean | std. dev. | mean | std. dev. | mean | std. dev. | mean | std. dev. |
| 0.05 | 0.049 | 0.004 | 0.103 | 0.017 | 0.428 | 0.010 | 0.125 | 0.012 | 0.446 | 0.012 | 0.423 | 0.011 |
| 0.1 | 0.098 | 0.006 | 0.140 | 0.014 | 0.407 | 0.009 | 0.181 | 0.018 | 0.412 | 0.021 | 0.335 | 0.015 |
| 0.2 | 0.204 | 0.007 | 0.198 | 0.009 | 0.383 | 0.018 | 0.251 | 0.018 | 0.366 | 0.022 | 0.225 | 0.007 |
| 0.3 | 0.296 | 0.017 | 0.216 | 0.010 | 0.342 | 0.007 | 0.295 | 0.042 | 0.363 | 0.037 | 0.173 | 0.017 |
| 0.4 | 0.406 | 0.013 | 0.235 | 0.005 | 0.342 | 0.014 | 0.321 | 0.023 | 0.337 | 0.029 | 0.121 | 0.013 |
| 0.5 | 0.499 | 0.006 | 0.240 | 0.005 | 0.378 | 0.011 | 0.299 | 0.014 | 0.323 | 0.015 | 0.093 | 0.010 |

TABLE IX
QUALITY MEASURES FOR DIFFERENT $r$ VALUES ($k = 10$) FOR THE BPI CHALLENGE 2017 LOG

| r | **Mean ratio** ($\mu$) | | **Std. Dev.** ($\sigma$) | | **Truly sampled** ($P_{\mathcal{T}}$) | | **Oversampled** ($P_{\mathcal{O}}$) | | **Undersampled** ($P_{\mathcal{U}}$) | | **Unsampled** ($P_{\mathcal{N}}$) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | mean | std. dev. | mean | std. dev. | mean | std. dev. | mean | std. dev. | mean | std. dev. | mean | std. dev. |
| 0.05 | 0.055 | 0.008 | 0.105 | 0.026 | 0.474 | 0.013 | 0.142 | 0.020 | 0.385 | 0.020 | 0.363 | 0.020 |
| 0.1 | 0.097 | 0.004 | 0.123 | 0.009 | 0.458 | 0.011 | 0.172 | 0.018 | 0.370 | 0.013 | 0.291 | 0.010 |
| 0.2 | 0.203 | 0.019 | 0.180 | 0.022 | 0.423 | 0.019 | 0.233 | 0.032 | 0.343 | 0.038 | 0.192 | 0.021 |
| 0.3 | 0.293 | 0.019 | 0.197 | 0.011 | 0.393 | 0.013 | 0.270 | 0.032 | 0.337 | 0.027 | 0.149 | 0.018 |
| 0.4 | 0.402 | 0.013 | 0.215 | 0.006 | 0.384 | 0.010 | 0.301 | 0.025 | 0.315 | 0.022 | 0.107 | 0.012 |
| 0.5 | 0.502 | 0.012 | 0.222 | 0.007 | 0.420 | 0.015 | 0.288 | 0.024 | 0.292 | 0.034 | 0.079 | 0.018 |

### A. The research setup

The event logs we use in our experiment are the BPI Challenge 2018 log [28] and the BPI Challenge 2017 log [29]. The 2018 log contains data of over 43,000 European Union subsidy applications, and was recorded in a period of three years and contains over 2M events. The 2017 log consists of 1,2M events and was collected from a Dutch financial institute in the years 2016 and 2017.

For the experiment, we created two plugins in ProM: one to sample the log and one to calculate the proposed measures. The first plugin requires an event log as input and returns an array of sampled logs as output. The plugin has two parameters: $k$ and $r$. Parameter $k$ determines how many samples are returned in the log array. Parameter $r$ determines the sample rate. For example, given $r = 0.5$ and $k = 10$, the plugin returns an event log array with 10 samples. Each sample, then, has approximately 50% of all the traces of the original log.

The second plugin requires the original event log, together with the output of the first plugin — the array of samples. The plugin calculates all quality measures and presents them to the user. It first computes the directly-follows frequency matrices of the original and all the samples. Next, the sample matrices are compared with the original matrix, creating $k$ sample ratio matrices. The behavioral quality measures proposed in Section III are then calculated for each matrix.

For the initial experiments, we restricted ourselves to $k = 10$ experiments per sample rate $r$. For $r$, we choose six values for $r$ ranging from $0.05$ to $0.5$. Below, we first inspect a single sample, after which the results of the complete experiments are reported.

### B. Inspecting a Single Sample

In Figure 5 the comparison of a sample with the original log is plotted. The sample was created from the BPI Challenge 2018 log with a sample rate of $r = 0.25$. The plot clearly shows that the tipping point of the graph centers around the expected ratio $r$ as theorized. As the plot depicts all possible behavior, not just the behavior of the event log, a large part is undefined. The unsampled behavior $\mathcal{N}$ is also clearly visible. This is an important class as it represents all the behavior that got lost due to sampling.

The data presented in Figure 5 can also be plotted in a histogram, as shown in Figure 6. This distribution shows the ratio on the horizontal axis and the frequency of that ratio on the vertical axis. The shape roughly resembles a normal distribution, although it is skewed towards the right side of the plot. Furthermore, there are peaks around the sample rate and at zero, both clearly distinguishable in the plot. These two points resemble the truly sampled behavior $\mathcal{T}$, and the unsampled behavior $\mathcal{N}$, respectively.

### C. Results

For the full experiment, we extracted quality measures for 6 values of $r$: $0.05$, $0.1$, $0.2$, $0.3$, $0.4$, and $0.5$. For each value for $r$, $k = 10$ samples were taken resulting in $6 \times 10 = 60$ samples for each experiment. All experiments were carried out with a true sample bandwidth of total size 0.05, resulting in an allowed distance of 0.025 from the sample rate. The results are shown in Table VIII and IX. For each quality measure both the mean and standard deviation are given over the 10 generated samples.

## V. DISCUSSION

In this Section, we discuss the results presented in Section IV. We also summarize relevant limitations of our study, and, last, a review of the potential threats to validity is included.

TABLE X
INFREQUENT BEHAVIOR IN THE BPI CHALLENGE 2018 LOG

| Frequency | Occurrence | Percentage of all behavior |
|---|---|---|
| $= 1$ | 80 | 13.47% |
| $\leq 2$ | 113 | 19.02% |
| $\leq 3$ | 147 | 24.75% |
| $\leq 5$ | 199 | 33.50% |
| $\leq 10$ | 252 | 42.42% |
| $\leq 20$ | 294 | 49.49% |

TABLE XI
INFREQUENT BEHAVIOR IN THE BPI CHALLENGE 2017 LOG

| Frequency | Occurrence | Percentage of all behavior |
|---|---|---|
| $= 1$ | 45 | 10.54% |
| $\leq 2$ | 73 | 17.10% |
| $\leq 3$ | 99 | 23.19% |
| $\leq 5$ | 116 | 27.17% |
| $\leq 10$ | 155 | 36.30% |
| $\leq 20$ | 199 | 46.60% |

### A. Findings - BPI Challenge 2018

A first observation is that the mean ratio $\mu$ is close to our $r$ values. Furthermore, the deviation from the mean ratio $\mu$ is small, implying that there was not too much variance among the samples. It is therefore worth noting that sampling a log this large will at the very least give the right sample ratio $\rho$ on average. The standard deviation $\sigma$, on the other hand, is very high when compared to the mean $\mu$, especially for lower values of $r$. For example, at $r = 0.05$, the standard deviation $\sigma$ is over twice the value of the mean. When taking larger samples by increasing the value for $r$ this effect reverses, with a standard deviation $\sigma$ about half the mean ratio $\mu$ at $r = 0.5$.

Both unsampled and undersampled behavior $\mathcal{U}$ steadily decrease when we increase the sample rate, although about 10% of the behavior in the original log is still lost at $r = 0.5$. It is interesting to note, however, that a large part of the original event log's behavior occurs infrequently (i.e. low values in the directly-follows frequency matrix). Table X shows the percentage of behavior that occurs less than a give number of times, excluding 0. These percentages explain at least a part of the high unsampled behavior $P_{\mathcal{N}}$. For example, unsampled behavior $P_{\mathcal{N}}$ for $r = 0.05$ is 0.423, while behavior with a frequency equal to or less than 20 is almost 50% of all log behavior. This suggests that there is a relation between low frequency behavior and relatively large values for $P_{\mathcal{N}}$.

Remarkably, the quality measure of undersampled behavior $\mathcal{U}$ is relatively stable. An explanation for this phenomenon is that unsampled behavior $\mathcal{N}$ increasingly becomes undersampled behavior $\mathcal{U}$ with higher $r$ values. The quality measure unsampled behavior $\mathcal{N}$ is therefore a useful extension of the undersampled behavior $\mathcal{U}$, as it provides insights in how behavior moves from unsampled to undersampled behavior.

### B. Findings - BPI Challenge 2017

The BPI Challenge 2017 log yielded results similar to the 2018 log, especially for the mean and standard deviation values. Just like the 2018 log, the standard deviation is about twice the mean for the lowest $r$ value, yet is only about half the mean for the highest value of $r$. Other similarities include the increase of oversampled behavior $\mathcal{O}$ and the decrease of the unsampled behavior $\mathcal{N}$ as $r$ increases, and the rather stable values for the undersampled behavior $\mathcal{U}$.

Infrequent behavior is shown in Table XI and occurs slightly less in the BPI Challenge 2017 event log, both absolute and relative. It seems that this affects the unsampled quality measure $\mathcal{N}$, as it is notably lower than the 2018 log, especially considering that the 2017 log has only half the number of events.

The similarity of the two sets of quality measures suggests that the behavioral quality of log samples follows certain patterns. For example, the true sample $\mathcal{T}$ slowly decreases when $r$ increases, yet from $r = 0.4$ to $r = 0.5$ it rises again. The oversampled behavior $\mathcal{O}$ shows a similar trend in which it rises with $r$, then starts decreasing at $r = 0.05$. However, more logs need to be analyzed before we can draw any definite conclusions. Furthermore, no definition or theory exists for defining what a 'good' or 'good enough' sample is. Our quality measures present the possibility to study this, but insight in and from the industry is also required.

Additionally, we found that the value for the true sample bandwidth affects the quality measures significantly. For example, at $r = 0.05$, the true sample bandwidth ranges from 0.025 to 0.075 ($0.05 \pm 0.025$). This leaves little room for undersampled yet not unsampled behavior ($\mathcal{U} - \mathcal{N}$), as seen in the results.

### C. Limitations

This study also has limitations. First, because we used the directly-follows relation as the basis for our research, it may not be applicable for algorithms that use other strategies. Next, because we tried to study the general behavior of log samples, it is quite possible that sampling techniques tailored specifically to a single algorithm perform better on that algorithm. Last, all conclusions in this research are based on the fact that we have the original log and therefore the complete directly-follows matrix. Event logs that are significantly different from the one evaluated here could yield different results, however, applying our method to more event logs would minimize generalization error.

### D. Threats to Validity

The data used in our research is from two well-known process mining challenges. This might introduce bias, as the data is selected specifically for this challenge by experts. Another important point for discussion is the way we sampled. This research uses completely random sampling, yet in practice true random sampling is unlikely due to bias, for example over time – as shown in the example used in Section III. However, our measures could determine sample quality for other approaches, such as stratified (random) sampling or non-probabilistic sampling, in the same way.

## VI. Conclusions and Future Work

In this paper we aim to address the potential problems with sampling large event logs. We propose six different quality measures to study the quality of samples, based on the directly-follows relation. The quality measures are implemented in ProM, and initial experiments have been carried out on real event logs. Our findings show the use of the proposed quality measures, as the experiments showed that samples vary widely in quality. As expected, the size of the sample plays a large role, but our experiments showed that even larger samples have quality issues. Furthermore, rare behavior gets lost easily and other behavior is often over or undersampled. More research is needed to validate our quality measures and confirm found patterns in sampling.

Several directions for future research are possible. Results of the initial experiments show that sample quality varies for different sample rates, and thus possibly the quality of the outcome after process mining analysis. Future work is to study the effect of sampling on model-based measures calculated on the result of process mining algorithms, and to compare these results with the quality measures we proposed in this paper.

The proposed measures depend on the original event logs. As such, we believe this is a first step towards statistical analysis of samples drawn from an infinite stream of events. The measures have the potential to apply and compare different sampling techniques, such as stratified sampling, on collecting relevant traces in event streams.

Last but not least, this paper proposes an initial set of measures to quantify the behavioral quality of log samples. More empirical research is required in analyzing sampling techniques. With the ever growing event logs, and complexity of process mining algorithms, more experiments on the quality of sampling are essential, in combination with case studies at organizations to validate their value in practice.

## References


[1] W. M. P. van der Aalst, *Process mining: data science in action*. Springer, 2016.

[2] B. F. van Dongen, A. K. A. de Medeiros, and L. Wen, "Process mining: Overview and outlook of petri net discovery algorithms," *Transactions on Petri Nets and Other Models of Concurrency II*, vol. 5460, pp. 225–242, 2009.

[3] W. M. P. van der Aalst, A. J. M. M. Weijters, and L. Maruster, "Workflow Mining: Discovering Process Models from Event Logs," *Knowledge & Data Engineering*, vol. 16, no. 9, pp. 1128–1142, 2004.

[4] J. M. E. M. van der Werf, B. F. van Dongen, C. A. J. Hurkens, and A. Serebrenik, "Process Discovery Using Integer Linear Programming," *Fundamenta Informatica*, vol. 94, no. 3 – 4, pp. 387 – 412, 2009.

[5] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Scalable process discovery and conformance checking," *Software and System Modeling*, vol. 17, no. 2, pp. 599–631, 2018.

[6] A. K. A. de Medeiros, A. J. M. M. Weijters, and W. M. P. van der Aalst, "Genetic process mining: an experimental evaluation," *Data Min. Knowl. Discov.*, vol. 14, no. 2, pp. 245–304, 2007.

[7] A. J. M. M. Weijters and J. T. S. Ribeiro, "Flexible heuristics miner (FHM)," in *IEEE Symposium on Computational Intelligence and Data Mining*. IEEE, 2011, pp. 310–317.

[8] J. C. A. M. Buijs and H. A. Reijers, "Comparing business process variants using models and event logs," in *EMMSAD 2014, Held at CAiSE 2014*, ser. LNBIP, vol. 175. Springer, 2014, pp. 154–168.

[9] W. M. P. van der Aalst, A. Adriansyah, and B. F. van Dongen, "Replaying history on process models for conformance checking and performance analysis," *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 182–192, 2012. [Online]. Available: http://dx.doi.org/10.1002/widm.1045

[10] A. Bolt and W. M. P. van der Aalst, "Multidimensional process mining using process cubes," in *Enterprise, Business-Process and Information Systems Modeling*, ser. LNBIP, vol. 214. Springer, 2015, pp. 102–116.

[11] L. Ghionna, G. Greco, A. Guzzo, and L. Pontieri, "Outlier detection techniques for process mining applications," in *International symposium on methodologies for intelligent systems*, ser. LNCS, vol. 4994. Springer, 2008, pp. 150–159.

[12] S. J. van Zelst, B. F. van Dongen, and W. M. P. van der Aalst, "Online discovery of cooperative structures in business processes," in *On the Move to Meaningful Internet Systems: OTM 2016*, ser. LNCS, vol. 10033. Springer, 2016, pp. 210–228.

[13] D. Fahland and W. M. P. van der Aalst, "Model repair - aligning process models to reality," *Inf. Syst.*, vol. 47, pp. 220–243, 2015.

[14] H. M. W. Verbeek, J. C. A. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst, "XES, XESame, and ProM 6," in *Information System Evolution*, vol. 72. Springer, 2011, pp. 60–75.

[15] R. van Langerak, J. M. E. M. van der Werf, and S. Brinkkemper, "Uncovering the runtime enterprise architecture of a large distributed organisation," in *International Conference on Advanced Information Systems Engineering*. Springer, 2017, pp. 247–263.

[16] W. M. P. van der Aalst, A. Adriansyah, A. K. A. de Medeiros *et al.*, "Process mining manifesto," in *Business Process Management*, ser. LNBIP, vol. 99. Springer, 2011, pp. 169–194.

[17] J. Carmona and J. Cortadella, "Process mining meets abstract interpretation," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2010, pp. 184–199.

[18] J. Carmona and R. Gavalda, "Online techniques for dealing with concept drift in process mining," in *International Symposium on Intelligent Data Analysis*. Springer, 2012, pp. 90–102.

[19] R. Conforti, M. La Rosa, and A. H. ter Hofstede, "Noise filtering of process execution logs based on outliers detection," Queensland University of Technology, Tech. Rep., 2015. [Online]. Available: https://eprints.qut.edu.au/82901/

[20] A. Berti, "Statistical sampling in process mining discovery," in *International Conference on Information, Process, and Knowledge Management*. IARIA, 2017, pp. 41–43.

[21] M. Bauer, A. Senderovich, A. Gal, L. Grunske, and M. Weidlich, *How Much Event Data Is Enough? A Statistical Framework for Process Discovery*. Springer, 2018, pp. 239–256.

[22] K. M. van Hee, Z. Liu, and N. Sidorova, "Is my event log complete? – a probabilistic approach to process mining," in *International Conference on Research Challenges in Information Science*. IEEE Computer Society, 2011, pp. 1–12.

[23] H. Y. J. W. J. Pei, L. Wen and X. Ye, "Estimating global completeness of event logs: A comparative study," *IEEE Transactions on Services Computing*, 2018, to appear.

[24] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Scalable process discovery and conformance checking," *Software & Systems Modeling*, vol. 17, no. 2, pp. 599–631, 2018.

[25] M. Bozkaya, J. Gabriels, and J. M. E. M. van der Werf, "Process diagnostics: A method based on process mining," in *International Conference on Information, Process, and Knowledge Management*. IEEE Computer Society, 2009, pp. 22–27.

[26] M. L. van Eck, X. Lu, S. J. J. Leemans, and W. M. P. van der Aalst, "PM$^2$ : A process mining project methodology," in *Advanced Information Systems Engineering*, ser. LNCS, vol. 9097. Springer, 2015, pp. 297–313.

[27] J. E. Freund, *Mathematical Statistics with Applications*. Pearson Education, 1962.

[28] B. F. van Dongen and F. Borchert, "BPI Challenge 2018," 2018. [Online]. Available: https://data.4tu.nl/repository/uuid:3301445f-95e8-4ff0-98a4-901f1f204972

[29] B. F. Van Dongen, "BPI Challenge 2017," 2017. [Online]. Available: https://data.4tu.nl/repository/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b

# Applying the Method of Reflections through an Event Log for Evidence-based Process Innovation

Pavlos Delias*†, Mehdi Acheli†, Daniela Grigori†
*Eastern Macedonia and Thrace Institute of Technology, Kavala, Greece
pdelias@teiemt.gr
†Univ. Paris-Dauphine, PSL Research University, CNRS UMR[7243], LAMSADE, 75016 Paris, France
{mehdi.acheli, daniela.grigori}@dauphine.fr

***Abstract*—Flexibility and adaptability have been praised and empirically validated as critical for an organization's performance. They are associated with the number of different behaviors that the organization exhibits, as well as with their level of sophistication, which in turn implies the number and the quality of the available knowledge and capabilities in place.**

**In this work, we propose the creation of a bipartite network, involving organizations and behaviors, through an event log, to capture the prospects of both organizations and behaviors for process innovation. By using the *Method of Reflections*, we claim that the structure of that network can expose a set of sophistication metrics, that eventually suggest *i)* the potential of behaviors for better organizational performance; *ii)* the most reachable innovation paths; *iii)* an "order of significance" over the behaviors considering their improvement contribution power.**

**We show the practical potential of our approach through a Proof of Concept based on a real dataset.**

***Index Terms*—Process Innovation, Bipartite Network, Evidence-based Decision Support**

## I. Introduction

As BPM (Business Process Management) discipline matures, its capabilities are expanding, reaching the state of the ambidextrous BPM, which refers to both the exploitation side ("the reliable execution of processes"), as well as to the exploration side ("the identification of opportunity points in processes") [1]. Towards the exploratory side of the spectrum, Recker [2] coined the term "Evidence-based process innovation" to emphasize the role that data (facts) have to play in process improvement.

In this work, we carry on this thread by trying to respond to a central research question: "How can evidence (in the form of an event log) support the innovation process of a company?". We claim that the proposed approach exemplifies a response to that question along three dimensions:

- By building awareness for process behaviors that contribute to better performance (thus reducing the process innovation latency).
- By facilitating change through recommendation for the most feasible innovation paths (thus mitigating organizational resistance).
- By suggesting innovations' prioritization considering their performance improvement potentials.

To reach the above contributions, we presume that there are several process behaviors (e.g., batch processing, involving multiple resources, balancing workload) that are routinely followed, or to which the company often resorts to run the process. In the case of low-structured processes, many such behaviors could be expected. Indeed, as it has been early identified [3], [4], in the quest for better performance, organizations will probably "pursue ambidexterity by creating a behavioral context that requires the integration of different activities at lower organizational levels" [5].

At these lower organizational levels, flexibility has been praised and empirically validated as critical for the firm's performance [5], [6]. Nevertheless, this flexibility cannot be directly linked to any particular capabilities or knowledge. It has been described in the literature either as a result of interconnected human resources practices (training, appraisal, etc.) [7], or as an aftereffect of the coordination of knowledge resources over the firm's social structures [8]. By accepting the definition of [9] about capabilities, i.e., that "*an organizational capability refers to the ability of an organization to perform a coordinated set of tasks, utilizing organizational resources, for the purpose of achieving a particular end result*", it becomes impractical to identify all the relevant capabilities for every behavior. So, if there is an imperceptible set of capabilities that affects performance, how can an organization get insights about them?

Even if capabilities cannot be observed, they can still be indirectly anticipated. Let us assume a tripartite network, the three node sets being the capabilities, the organizations, and the behaviors (patterns). Organizations are connected to their available capabilities, and patterns are connected to their required capabilities. Then, if a pattern is observed, we can assume that all the required capabilities are in place, therefore, we can represent the tripartite network using a bipartite one, connecting organizations to patterns. By observing the patterns that are exhibited in each organization, we can get valuable insights about the capabilities that it possesses. In this context, we find the Method of Reflections (MoR) [10] particularly relevant, since MoR claims that the characterization of the structure of the bipartite network can illustrate the allocation of the hidden layer (in our case the capabilities) to the left set of nodes (in our case the organizations).

MoR, originally proposed as a view of countries' economic growth and development, calculates economic complexity through a bipartite network of countries-products, and suggests

978-1-7281-0919-0/19/$31.00 ©2019 IEEE
DOI 10.1109/ICPM.2019.00025

that economic complexity is correlated to a country's level of income. The basic variables that MoR deals with are: diversity of countries (how many different products a country exports); ubiquity of products (how many countries export that product); product complexity (how complex is a product); economic complexity (how complex is the economy of a country).

In this paper, we develop an analogy between the original MOR context (country development) and process innovation. We propose a bipartite network model and show how the variables that MoR computes on this network can be interpreted as Process Innovation relevant variables, centered around what we call as *"Operations Sophistication"* (OS). The intuition behind the Operations Sophistication can be described as follows.

As we have discussed in the previous paragraphs, knowledge and capabilities are reflected to the level of flexibility and adaptability of an organization (we shall call this composite virtue as versatility from here on). However, the fact that knowledge is tacit and that the capabilities' connections to patterns are rather vague, makes very difficult, if not impossible, to identify the tacit knowledge and the capabilities required to improve the performance of organizations.

Therefore, we claim that the performance of an organization resides in the diversification of its available capabilities (versatility). Sophistication is expressed in the conglomeration of the exhibited patterns and reflects the mechanisms that are needed to exploit different sets of capabilities. A more versatile organization is expected to be able to demonstrate higher sophistication, apply more complex patterns, and hence achieve better performance. The analogies with MoR variables are therefore the following:

- Versatility reflects MoR's diversity of products. In MoR, we want countries to export as many different products as possible, here we want organization to demonstrate as many desired patterns as possible (and avoid as many undesired patterns as possible too).
- Pervasiveness of patterns is defined similarly to ubiquity of products.
- Operations Sophistication stands for a rank of organizations based on how versatile and sophisticated their pattern behavior is, hence, it corresponds to MoR's economic complexity. The core assumption of this work is that the favored situation for an organization is to possess a high operation sophistication index, like the favored situation for countries is to have a high economic complexity index.
- Patterns Sophistication ranks the sophistication of patterns. We assume that more sophisticated patterns reflect a more complex set of capabilities, so an organization that would possess patterns with higher sophistication has an advantage over an organization that possesses less sophisticated patterns. This is exactly the intuition of the product complexity in MoR.

The methodology to measure Operations Sophistication, together with more detailed justifications, are presented in section II. Then, in section III we illustrate our approach with a proof of concept, based on real data. Finally, a short discussion concludes the paper.

## II. Methodology

In this section we present the proposed methodology for applying MoR for evidence-based process innovation, whose steps are described in Workflow 1.

The starting point of our work is an event log [11, p. 128]. We have to stress that this is what we mean in this work when we refer to "evidence". Arguably, the richer the event log in terms of variables, the greater the potentials for insights to be discovered.

**Workflow 1:** Methodology steps for operations-sophistication-based Process Innovation

**Input :** An event log $\mathcal{L}$, a set of organizations $\mathcal{O}$,a threshold $a$ for the *Authentic Pattern Adoption*

1 **Stage** *1: Event Log Transformations*
2 Define a performance metric $\mathcal{P}$;
3 Discover a set of case-wise patterns $\mathbf{P}$ through $\mathcal{L}$;
4 Create a $cases \times patterns$ matrix *CP*;
5 Characterize each $p \in \mathbf{P}$ as *desired / non-desired* with respect to $\mathcal{P}$;
6 Convert *CP* into a binary matrix *BM*;
7 Sum the rows of *BM* per $o \in \mathcal{O}$ to create a frequency $|\mathcal{O}| \times |\mathbf{P}|$ matrix *M*;
8 **foreach** $M_{op}$ **do**
9 Calculate the *Strong Pattern Adoption* of the organization $o$ in pattern $p$ $SPA_{op}$;
10 **end**
11 Calculate a binary matrix $\mathbf{B}_{op}$ by comparing $SPA_{op}$ to $a$;
12 **return** *The binary matrix* $\mathbf{B}_{op}$*;*
13 **Stage** *2: Getting Sophistication Metrics*
14 Calculate *Versatility* $\forall o \in \mathcal{O}$;
15 Calculate *Pervasiveness* $\forall p \in \mathbf{P}$;
16 **return** *Operations Sophistication Index, Patterns Sophistication Index*;
17 **Stage** *3: Calculating Opportunity Gain*
18 **foreach** $p, p' \in \mathbf{P}$ **do**
19 Calculate *proximity* $\phi_{pp'}$;
20 **end**
21 **foreach** $o \in \mathcal{O}, p, \in \mathbf{P}$ **do**
22 Calculate *distance* $d_{op}$;
23 **end**
24 Calculate *Opportunity Value* $\forall o \in \mathcal{O}$;
25 **foreach** $o \in \mathcal{O}, p, \in \mathbf{P}$ **do**
26 Calculate *Opportunity Gain*;
27 **end**
28 **return** *proximity, distance,Opportunity Value, Opportunity Gain*;

### A. A Bipartite Network for Process Innovation

The first stage of the methodology (line 1 in Workflow 1) is about mining the event log to construct the bipartite network

represented as a binary matrix on which the MOR method will be applied. First the analyst defines a performance metric (line 2), i.e., the dimension under which the effectiveness of the process execution is observed, described, measured.

The next step is to discover case-wise elements of the process behavior (patterns) from the event log (line 3). A pattern can be anything related to how the case is operated. For example, a pattern can be the number of handovers that are taking place, the batch processing, the frequency of activities per weekday, anything that could potentially affect the performance metric for a case. There are no limitations for the number of patterns that are required. We recommend following a maximalistic attitude in discovering patterns, since the structure of the network will reveal their characteristics during the subsequent phases. However, one has to worry about the bias in patterns discovery, namely, patterns around a particular feature should not be over-represented. An additional warning concerns the face validity of the patterns, namely, if each pattern appears to be a legitimate measure of the performance metric. Without any doubt, a critical limitation in this approach is that there are no guarantees that the patterns that will be discovered will be complete, i.e., that they will exhaustively reflect the operations' culture and reality. Finally, a last note about patterns is that we can use different scales to measure them (e.g., absolute numbers or percentages). After patterns discovery, a matrix is built whose rows are the cases, whose columns are the patterns, and where the values at the cells reflect how much every pattern occurs in every case (line 4).

The next step is to assess the contribution of every pattern to the performance metric (by classifying it either as desired or as undesired - line 5). Such a flag can be assessed by expert judgment, but since the underlying paradigm of this work is the evidenced-based approach, we suggest assessing it through a numerical method. A simple correlation of the pattern with the given performance metric can be used.

Then the $cases \times patterns$ matrix is transformed into a binary matrix (line 6). This step is required in order to have uniform representations for patterns that can be measured with different scales and to differentiate between organizations that are exhibiting or not the pattern.

A straightforward way to do this is by setting a threshold. Such a threshold can be the average value, but other alternatives can be used without having to change anything in the workflow of the method. For cases that are above the mean value in a particular pattern, we put the value 1 at the corresponding cell of the matrix, or the value 0 otherwise. This happens for the desired patterns, while for the undesired ones, the opposite holds. This step is crucial for consistency. In the original MoR, it is self-evident that the more products a country exports, the better for its economy. In our context, this does not hold for every pattern. On the contrary, we anticipate that some patterns could harm the performance metric. In this case, the desired situation for an organization is to avoid them, therefore, the actual pattern that would be included in the binary matrix is the *avoidance* of the non-desired pattern. This way, the *versatility* metric, which suggests "the more patterns an organization exhibits, the better" becomes consistent. At this step, we will end with a binary matrix (0,1), which reflects which cases are exhibiting which patterns.

The next step is to summarize that matrix per organization (line 7). This will return a matrix which will count the frequencies of the cases per organization that are demonstrations of the corresponding patterns. However, in order to proceed, we need to account for the number of cases that are performed by organizations as well as for the commonness of the patterns. This need exists because large organizations (with more cases) are expected to present higher counts for the same patterns, as well as because some patterns are expected to occur more frequently than others across all organizations.

To this end, we use the Balassa index for Revealed Competitive Advantage [12]. Balassa' definition, originally introduced in the context of international trade, assumes that a country has a "*revealed competitive advantage*" if it exports more than its "fair" share, i.e., its national share of exports of a product is larger than the world share of that product in global trade. In our context, this index does not actually reveal a "competitive advantage" but rather an organization that substantially applies the pattern, therefore we use the label "Strong Pattern Adoption" (line 9). If $\boldsymbol{M}$ is a matrix which presents the frequencies of every organization in every pattern, then the strong adoption of an organization $o$ in pattern $p$ is given by

$$SPA_{op} = \frac{\boldsymbol{M}_{op}}{\sum_o \boldsymbol{M}_{op}} \Big/ \frac{\sum_p \boldsymbol{M}_{op}}{\sum_{o,p} \boldsymbol{M}_{op}} \tag{1}$$

This metric will guide the construction of our bipartite network. In particular, if a pair (organization, pattern) has a corresponding $SPA_{op}$ greater or equal to a threshold $a$ (typically close to 1), we put a connection on the network, else we assume that there is no such connection (line 11). Formally, the network can be represented by a matrix $\boldsymbol{B}$, where

$$\boldsymbol{B}_{op} = \begin{cases} 1, & \text{if } SPA_{op} \geq a \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

### *B. Measuring Operations Sophistication*

The basic assumption in our work is that if exhibiting a pattern requires a particular (yet unknown) combination of knowledge and capabilities, then it is implied that the organizations which are actually featuring that pattern possess the required knowledge and capabilities. This assumption allows a reasoning that leads to the construction of metrics for the operations sophistication (stage 2 in Workflow 1).

Considering that a versatile organization can exploit a larger set of capabilities to adopt to many different business requirements, we define *versatility* as the number of different patterns that an organization demonstrates. In addition, we define the *pervasiveness* of a pattern as the number of organizations that demonstrate it. We expect that patterns that require complex combinations of capabilities to be less pervasive. Versatility and pervasiveness can be both considered as approximations of the number and of the variety of capabilities. Both are affected

by the existence of rare capabilities (e.g., an expertise in a very specific domain). However, it is possible to tell if low pervasiveness is due to oddity or sophistication, by observing the number of the other patterns that the organizations which exhibit the rare patterns are able to exhibit too. If those organizations are only exhibiting few other patterns, then probably, the low pervasiveness is due to oddity. If they are exhibiting a plethora of patterns, probably the low pervasiveness of the pattern is due to its sophistication, because it reflects a complex combination of capabilities in place.

By the same token, versatility can be used to correct pervasiveness, as well as pervasiveness can be used to correct subsequent estimations of versatility. This is exactly the essence of MoR which "consists of iteratively calculating the average value of the previous-level properties of a node's neighbors" in a bipartite network [10]. In the following, we present how MoR can be applied in our context.

First, we need to express versatility and pervasiveness in terms of the bipartite network, or else, through the matrix $\boldsymbol{B}_{op}$ that was introduced by (2). Following the descriptions in the above paragraphs, versatility and pervasiveness can be defined as the sum of the rows and the columns of that matrix respectively (lines 14-15). So, let

$$Versatility := \boldsymbol{v}_{o,0} = \sum_{p} \boldsymbol{B}_{op} \tag{3}$$

and

$$\textit{Pervasiveness} := \boldsymbol{p}_{p,0} = \sum_{o} \boldsymbol{B}_{op} \tag{4}$$

Formulas (3) and (4) are actually calculating the degrees of the nodes in the bipartite network, considering the partition they belong to (organizations or patterns) and act like the initial conditions for the recursive formulas that we will define next. Since we regard versatility and pervasiveness to be approximations of the number of capabilities (that an organization possesses, or that a pattern requires respectively), we need to correct those initial metrics by using each one to correct the other. According to MoR, for organizations, this means calculating the average pervasiveness of the patterns they exhibit (during the first iteration), the average versatility of the organizations that are exhibiting those patterns (during the second iteration), and so on. Similarly, we shall act for patterns (iteratively calculating the average value of the previous-level). The corresponding recursive equations are:

$$\boldsymbol{v}_{o,N} = \frac{1}{\boldsymbol{v}_{o,0}} \sum_{p} \boldsymbol{B}_{o,p} \cdot \boldsymbol{p}_{p,N-1} \tag{5}$$

$$\boldsymbol{p}_{p,N} = \frac{1}{\boldsymbol{p}_{p,0}} \sum_{o} \boldsymbol{B}_{o,p} \cdot \boldsymbol{o}_{o,N-1} \tag{6}$$

To give an intuition of those metrics, let us think their first iteration: Both $\boldsymbol{v}_{o,1}$ as well as $\boldsymbol{p}_{p,1}$ are just expressions of the network degrees of their neighbors, a metric which in network analysis is defined as the average nearest neighbor degree [13]. Table I presents how the metrics of the first three levels can be interpreted.

TABLE I
INTERPRETATION OF THE SOPHISTICATION METRICS FOR THE FIRST ITERATIONS

| Metric | Description |
|---|---|
| $\boldsymbol{v}_{o,0}$ | Versatility. Number of patterns exhibited by organization $o$ |
| $\boldsymbol{p}_{p,0}$ | Pervasiveness. Number of organizations exhibiting pattern $p$ |
| $\boldsymbol{v}_{o,1}$ | Average pervasiveness of the patterns exhibited by organization $o$ |
| $\boldsymbol{p}_{p,1}$ | Average versatility of the organizations exhibiting pattern $p$ |
| $\boldsymbol{v}_{o,2}$ | Average versatility of organizations with a pattern profile similar to organization $o$ |
| $\boldsymbol{p}_{p,2}$ | Average pervasiveness of the patterns exhibited by organizations that exhibit pattern $p$ |

It is also very interesting to observe what happens if we keep the recursion of those equations. By inserting (6) into (5) we get

$$\begin{aligned} \boldsymbol{v}_{o,N} &= \frac{1}{\boldsymbol{v}_{o,0}} \sum_{p} \boldsymbol{B}_{op} \frac{1}{\boldsymbol{p}_{p,0}} \sum_{N(o)} \boldsymbol{B}_{N(o)p} \cdot \boldsymbol{v}_{N(o),N-2} \\ &= \sum_{N(o)} \boldsymbol{v}_{N(o),N-2} \sum_{p} \frac{\boldsymbol{B}_{op}\boldsymbol{B}_{N(o)p}}{\boldsymbol{v}_{o,0}\boldsymbol{p}_{p,0}} \end{aligned} \tag{7}$$

where $N(o)$ is the set of neighbors of $o$. By setting

$$\widetilde{\boldsymbol{B}}_{oN(o)} = \sum_{p} \frac{\boldsymbol{B}_{op}\boldsymbol{B}_{N(o)p}}{\boldsymbol{v}_{o,0}\boldsymbol{p}_{p,0}} \tag{8}$$

equation (7) can be rewritten as:

$$\boldsymbol{v}_{o,N} = \widetilde{\boldsymbol{B}}_{oN(o)} \times \boldsymbol{v}_{o,N-2} \tag{9}$$

which when $N \to \infty$ takes the shape of:

$$\widetilde{\boldsymbol{B}} \times \boldsymbol{v} = \lambda \boldsymbol{v} \tag{10}$$

which means that $\boldsymbol{v}$ is an eigenvector of $\widetilde{\boldsymbol{B}}$. What we are interested in is what explains the variance in the system, so we take the eigenvector that corresponds to the second largest eigenvalue (the vector of the largest eigenvalue is expected to be a replication of the same value) and, following the MoR, we define it as the *operations sophistication index* (OSI) for our organizations (line 16). In particular, OSI is the normalized value of the second largest eigenvector.

We can work the same way to derive a *patterns sophistication index* (PSI), which will be calculated through a formula similar to (9) but instead of the metric $\boldsymbol{v}_o$ we will have $\boldsymbol{p}_p$ and the $\widetilde{\boldsymbol{B}}$ matrix will be replaced by a new one

$$\widetilde{\boldsymbol{B}}_{pN(p)} = \sum_{o} \frac{\boldsymbol{B}_{op}\boldsymbol{B}_{oN(p)}}{\boldsymbol{v}_{o,0}\boldsymbol{p}_{p,0}} \tag{11}$$

### C. Implications of the Sophistication Metrics

An important implication of the model that we presented in sections II-A and II-B comes from the fact that patterns have some kind of similarity to each other. Indeed, that follows from our basic assumption that patterns emerge due to available capabilities. Since we cannot directly observe capabilities, we have to resort again to an indirect measure. In [14], authors present how such an indirect similarity metric can be derived

through a bipartite network. The basic idea is that nodes of the one part (patterns) that are connected to the same instances of the nodes of the other part (organizations) imply that the latter nodes (organizations) share the requirements to exhibit the patterns. Therefore, they introduce the metric of *proximity* $\phi$ between any pair of one part's nodes (patterns) as the probability that those patterns co-occur in different nodes of the other part (organizations). In other words, if an organization exhibits one pattern, then, it is more probable that it will also exhibit other patterns that are "proximate" to the first one rather than patterns that are "distant" to it (line 19).

This probability is actually a conditional one, which measures the probability that an organization exhibits pattern $p$, given that it exhibits pattern $p'$. The problem with that approach is that the derived metric is not symmetric (hence, not suitable for capturing similarity). To address this issue, authors of [14] propose to use the minimum conditional probability of $p|p'$ and $p'|p$ and assign it to both pairs. Following our notation, the proximity between two patterns $p$ and $p'$ can be calculated directly through the bipartite network matrix and it will be

$$\phi_{pp'} = \frac{\sum_o \boldsymbol{B}_{op}\boldsymbol{B}_{op'}}{\max\left(\boldsymbol{p}_{p,0}, \boldsymbol{p}_{p',0}\right)} \tag{12}$$

Proximity $\phi$ measures the similarity between patterns. However, it would be also very informative to be able to measure an analogous metric between organizations and patterns. Ideally, we would like such a metric to return a low distance between an organization $o$ and a pattern $p$ when the organization exhibits most of the patterns that are proximate to $p$, and the inverse (line 24). By interpreting the distance metric that it is proposed in [15], we can create such a metric $d_{op}$ by summing the proximities of the patterns that are not yet exhibited by $o$, normalized over the sum of the proximities of all patterns (see (13) )

$$d_{op} = \frac{\sum_{p'} (1 - \boldsymbol{B}_{op'})\phi_{pp'}}{\sum_{p'} \phi_{pp'}} \tag{13}$$

Distance $d$ is an informative metric that can be calculated through the Bipartite Network matrix. Additional information that is carried by the sophistication metrics is not taken into account. Such information could bring more insights about organizations, since it would indicate not only if there are many unexploited patterns at close distance (thus more easily reachable), but if those patterns are sophisticated as well, revealing the sophistication prospects of the organization. Again, following [15], we can call this the *opportunity value* of an organization (line 24) and calculate it as:

$$OV_o = \sum_{p'} (1 - d_{op'})(1 - \boldsymbol{B}_{op'})PSI_{p'} \tag{14}$$

Last, in order to be able to respond to the third question we presented in the introduction, i.e., to suggest a prioritization over the patterns, we need to find the impact that there will be for an organization $o$ to its opportunity value, assuming it will adopt pattern $p$. This metric is called *opportunity gain* (line 26) and can be calculated as:

$$OG_{op} = \sum_{p'} \frac{\phi_{pp'}}{\sum_{p''} \phi_{p''p'}}(1 - \boldsymbol{B}_{op'})PSI_{p'} - (1 - d_{op})PSI_p \tag{15}$$

## III. Proof of Concept

### A. Data and Case Description

To demonstrate the practical potential of our approach, we used the data from Business Process Intelligence Challenge 2015 (BPIC 2015) workshop [16], where five Dutch municipalities opened their data to invite evidence-based insights about their building permits process. The original dataset describes 262628 events that correspond to 5649 cases (applications), and it covers a period of approximately five years (from 2010 to 2015). Every row of the dataset registers the case it refers to, the actual activity that takes place (e.g., creating cover letter decision), the time-stamp of the event, the resource that performs the activity, and many other relevant attributes (e.g., the type of the building). Before making any analysis, we pre-processed the log file like it is described in [17]. More specifically, to work with homogeneous cases, we filtered the original dataset and kept just the cases that applied for building (Bouw in Dutch). Moreover, to avoid the bias of still open cases, we accepted just the cases that have reached an end status. Finally, we removed cases that were longer than 2 years (a step not included in [17]). The pre-processed dataset consists of 103739 events that correspond to 2044 building permit applications (completed cases).

### B. The Bipartite Network

In this section we exemplify our approach by creating a bipartite network where organizations exist in the one part and patterns in the other. The data we described in section III-A contain already 5 different organizations (5 municipalities). However, this number is rather small, considering the technicalities we presented in section II. So, to demonstrate our method, we created artificial organizations by splitting the municipalities' cases per year (the cases that spanned more than 1 year were allocated to the year when they started). We acknowledge the limitations of this handling, since organizations are not distinct entities this way, therefore we do not claim the factual value of the results - we emphasize that section II-C is a "Proof of concept" used for demonstration reasons. Anyway, splitting by year resulted in 30 organizations (6 facets per municipality). These comprise the set $\mathcal{O}$ (see *Input* in Workflow 1).

The first step (line 2) is to assume that performance is equivalent to the cases' duration, the shorter the better. Moving forward (lines 3-4 of Workflow 1), we consider the patterns that we present in Table II. Patterns no. 5, 7-9 were calculated according to the corresponding implementations of [18]. The time window that is referred in patterns 4 & 6 was set to 3 weeks. We shall notice that all patterns are calculated at the case level. To decide if a pattern is a desired

TABLE II
DISCOVERING PATTERNS OF PROCESS BEHAVIOR.

| Pattern | Description |
|---|---|
| 1.Assignation | Number of resources involved in case. |
| 2.Sociability | Average number of collaborators per resource involved. For each resource, the number of collaborators is calculated organization-wise. |
| 3.Practised | Average number of cases where each participating resource is involved organization-wise. |
| 4.Multitasking | Average number of cases where each participating resource is involved during a specified time window. |
| 5.Specialization | Average number of distinct tasks performed by each involved resource. For each resource, the number of distinct tasks is calculated organization-wise. |
| 6.Workload | Average number of activities performed by each resource during a specified time window. |
| 7.Repeats | Number of tasks that were executed not immediately following each other by the same resource. |
| 8.Redos | Number of tasks that were executed not immediately following each other by different resources. |
| 9.Selfloops | Number of tasks that were executed immediately following each other by any resources. |
| 10.Handovers | Number of switches between different resources for two consecutive activities. |
| 11.Ping-Pong | Number of Ping-Pongs. A Ping-Pong takes place when a resource is revisited during a case, unless this happens during consecutive events. |
| 12.Batch processing | Number of times that the case was involved in batch processing. We assume that a batch processing happens if on a single day, for a single organization, any activity is performed for more than two cases. |
| 13.Weekdays variation | The standard deviation for the number of activities performed per weekday. |
| 14-44. Phases | Percentage of activities spent in each phase. Phases were identified by the original activity codes. Each activity code consists of three parts: two digits, a variable number of characters, and then three digits. A phase was defined as the first two parts plus the first digit of the third part. 42 phases were identified, but 11 of them were removed, because they were rarely visited. |

or an undesired one (line 5), we empirically observed its correlation with the cases' duration (the selected performance metric). Interestingly, the only patterns that turned out to be desired were the "workload" and phases "01_HOOFD_0", "01_HOOFD_3", "01_HOOFD_4", "01_HOOFD_5", "02_DRZ_0", "09_AH_I", and "13_CRD_0". Then, it is quite straightforward to apply eq. (1) and (2) to execute the remaining lines of stage 1 of the Workflow.

However, we ought to discuss here a critical issue about patterns' discovery. The set we presented is not complete, and of course, it is subject to the event log richness. For example, if two time-stamps are available per event, far more informative patterns can be discovered. This "completeness" issue is therefore an inherent limitation. Yet, our thesis is that it is not realistic to expect discovering absolutely all relevant patterns that affect the process performance. However, our point of view is that this inherent limitation should not be a prohibitive factor for the application of the method, neither it should lead to a frenzied quest of patterns discovery, but that it should be treated and well-respected during the results interpretation phase. We cannot stress enough that the results of our approach should always be subject to the process stakeholders' judgment - they are recommendations and not automatic solutions.

### C. Results

As we have discussed in the previous sections, *Operations Sophistication* reflects the knowledge and capabilities that an organization possesses and the availability of the required mechanisms to exploit them. As such, it is important because it is a process innovation potential index. Moreover, in this work, we claim that it is additionally interesting because it relates to performance.

Since in this work we have assumed that performance is assessed through duration, in Fig. 1 we plot the average duration for every organization's cases against its OSI. The blue line is the respective regression line ($R^2 = 0.3023$, $P = 0.0009$) and the grey area around it is the $95\%$ confidence interval. We observe that there is a clear negative effect (as OSI falls, duration increases), although there is a lot of unexplained variation (a glimpse of the issue of completeness). Besides the unexplained variation, in this running example, we observe two peculiar deviations C.2011 and A.2011 which both have a low OSI but a good performance. There is an additional reason that these two organizations remarkably deviate. First, we should recall that C.2011 and A.2011 are "artificial" organizations that inherit from the real organizations, municipalities C and A. A and C are the overall top performing municipalities, therefore, we observe a kind of "ceiling effect",i.e., they began by being better and it may not have been possible to improve on this level of performance.

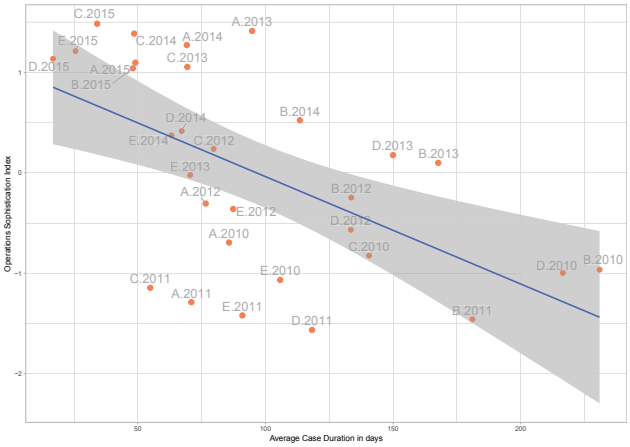



Fig. 1. Relationship between Operations Sophistication Index and average cases' duration for organizations

Overall, OSI is a sign not only for process innovation potential, but for favorable performance as well. In Table III we present the top and the bottom performing organizations in terms of OSI. The OSI of an organization is reasonably connected to the sophistication of the patterns that it demonstrates. In the long run, organizations can only increase their operations sophistication index by adhering to an increasing number of sophisticated patterns. This fits as a response to

the first research question we expressed in section I, namely "building awareness for process behaviors that contribute to better performance", since organizations should be more vigilant on patterns with a higher PSI. Fig. 2 illustrates the PSI for every pattern we considered in our case study.

TABLE III
OPERATIONS SOPHISTICATION INDEX. TOP AND BOTTOM PERFORMING ORGANIZATIONS.

| Top 5 Organizations | Bottom 5 Organizations |
|---|---|
| C.2015 | D.2011 |
| A.2013 | B.2011 |
| C.2014 | E.2011 |
| A.2014 | A.2011 |
| E.2015 | C.2011 |

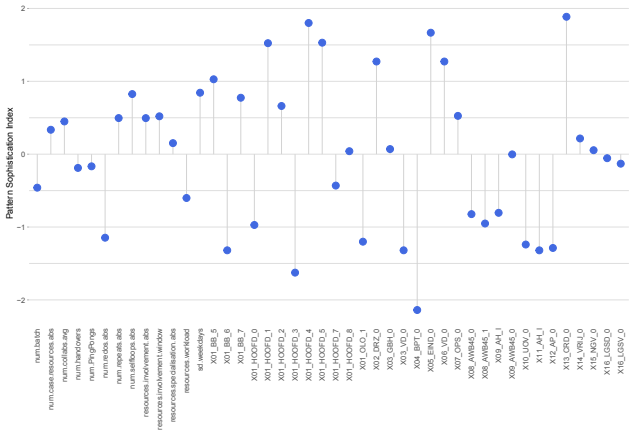



Fig. 2. Patterns Sophistication Index

The second contribution we claimed in section I was the highlighting of the most feasible innovation paths, namely the highlighting of the patterns that are more reachable. In our context, this takes the shape of the proximity between patterns (lines 18-20). More specifically, assuming an organization is already adopting a specific pattern, evidence suggests (based on co-occurrences, see section II-C) that it would be better-tempered in adopting the proximate patterns. To this end, we plot in Fig. 3 a network of a subset of patterns (we left out the patterns that refer to phases), which illustrates these (more feasible) paths. In that network, the size of a node is proportional to its PSI (i.e., organizations would benefit more if they move towards large nodes), while we have kept only the edges with a weight (proximity) larger than 0.6 to keep the graph uncluttered. For instance, if an organization is adhering to the pattern of batch processing (node "num.batch" in the network), then it would be more difficult to start pursuing the pattern of weekdays variation ("sd.weekdays") because there is no direct connection between those two in the network.

The network illustration provides recommendations about the more reachable innovation paths, however, those recommendations are not tailored to the individual needs of

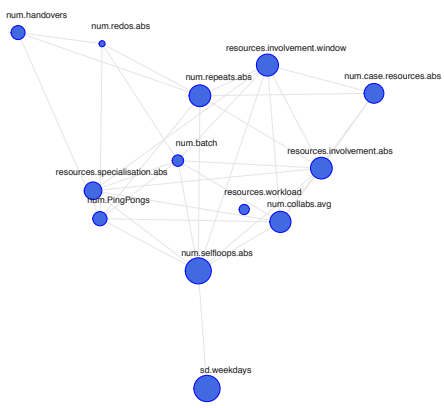



Fig. 3. Network of selected patterns based on their proximities

organizations. Nevertheless, with equation (15) (lines 21-28), which calculates the *opportunity gain* of every pattern for every organization, we hold this kind of information, so we can get even more insightful information, like the one depicted in Fig. 4. In that chart, every row is dedicated to an organization. Columns represent patterns, and the circles in the cells stand for the opportunity gain values. There is a colored spectrum (blue for large positive values, orange for large negative ones) that guides our recommendations for the OS improvement potentials for each organization (the size of the circle is an additional visual aid). Ultimately, by looking at such a chart, we are able to recommend to each organization what patterns have a greater capacity in improving its operations sophistication index, and hence, how it should challenge its process innovation prospects.

Although the pre-processing that we committed to the original dataset makes our results non directly comparable to other works that analyzed the same data, we observe a consensus view in several points. First, the winner of BPIC'15 [19] propounds that they have been major changes in May 2012. This coincides with our results that show that all organizations exhibit different behaviors before and after 2012 (see Fig. 4 as well as the vertical dimension in Fig. 1). Then, in [19], it is convincingly argued that handovers and specialization have little effect on performance. These claims seem to match what PSI suggests (see Fig. 2) since the patterns we defined with similar operationalizations (Handovers, Ping-Pong, and Specialization) have low PSI values. However, our results are rather different with respect to reworks. In [19] reworks are reported to have little effect. In this example, we distinguished three different types of reworks (repeats, redos, and selfloops) and we observed that they are affecting some organizations (see Fig. 4). Moreover, phases X01_BB_5 and X01_BB_7 (objections and appeals) are reportedly causing delays, something that our analysis confirmed by assigning a high PSI to their avoidance.

The added value of our approach is that we do not just return the contribution of each pattern to performance. We

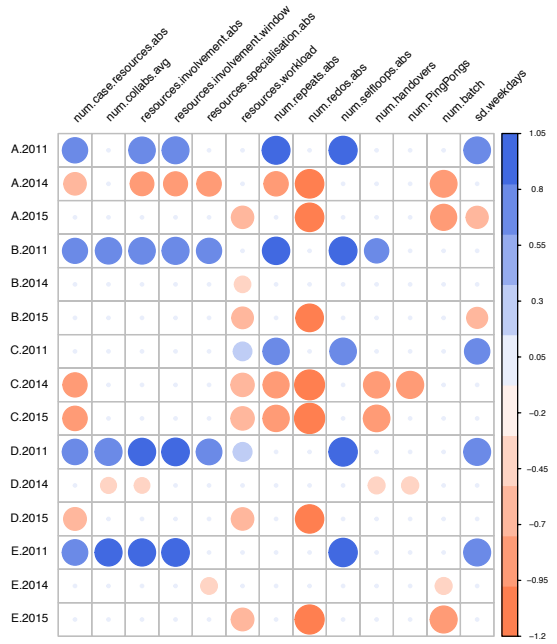



Fig. 4. Opportunity gain for selected organizations and patterns

provide a quantification of their contribution to process innovation, we diagnose the feasibility of their adoption (proximity network), as well as we reveal their opportunity gain for each organization. In addition, we do this by collectively examining the set of patterns and without requiring special, individual hypotheses for each pattern.

## IV. Discussion

Process improvement, for all its importance for organizations, can be reached through various approaches, ranging from incremental improvements to re-engineering, to process innovations that would require a more intrinsic rethinking. In this work, we presented an evidence-based approach that supports a well-informed rethinking for process improvement. We proposed a set of measures that capture the *operations sophistication* of an organization and provided suggestions of what are their implications for business. This work can also be used with a different target bipartite network. We foresee two options: While patterns will consist of a set of nodes in every case, the first option comprises the resources as the second set of nodes, and the second option comprises organizations that are performing the same process.

We should stress that this work is not focusing on the required capabilities and on how they can be acquired., but rather on how an event log can deliver insights to organizations about what low-level behaviors they should pursue. It is within our future plans to check how this method can function in a Resource-Patterns context (rather than the Organizations-Patterns context presented here), and to improve several marginal technicalities. This approach, of course, needs to be validated in a real setting, involving the actual process stakeholders, since it is of principal importance to convey the actuality that process innovation cannot be automated, yet it can benefit from the automation that process analytics provide.

## References


[1] M. Rosemann, "Proposals for Future BPM Research Directions," in *Asia Pacific Business Process Management*, W. van der Aalst, J. Mylopoulos, M. Rosemann, M. J. Shaw, C. Szyperski, C. Ouyang, and J.-Y. Jung, Eds. Cham: Springer International Publishing, 2014, vol. 181, pp. 1–15.

[2] J. Recker, "Evidence-Based Business Process Management: Using Digital Opportunities to Drive Organizational Innovation," in *BPM - Driving Innovation in a Digital World*, J. vom Brocke and T. Schmiedel, Eds. Cham: Springer International Publishing, 2015, pp. 129–143.

[3] C. Andriopoulos and M. W. Lewis, "Exploitation-exploration tensions and organizational ambidexterity: Managing paradoxes of innovation," *Organization Science*, vol. 20, no. 4, pp. 696–717, 2009.

[4] T. J. M. Mom, F. A. J. van den Bosch, and H. W. Volberda, "Understanding variation in managers' ambidexterity: Investigating direct and interaction effects of formal structural and personal coordination mechanisms," *Organization Science*, vol. 20, no. 4, pp. 812–828, 2009.

[5] P. C. Patel, J. G. Messersmith, and D. P. Lepak, "Walking the tightrope: An assessment of the relationship between high-performance work systems and organizational ambidexterity," *Academy of Management Journal*, vol. 56, no. 5, pp. 1420–1442, 2013.

[6] S. A. Way, P. M. Wright, J. B. Tracey, and J. F. Isnard, "Hr flexibility: Precursors and the contingent impact on firm financial performance," *Human Resource Management*, vol. 57, no. 2, pp. 567–582, 2018.

[7] I. Beltrn-Martn, V. Roca-Puig, A. Escrig-Tena, and J. C. Bou-Llusar, "Human resource flexibility as a mediating variable between high performance work systems and performance," *Journal of Management*, vol. 34, no. 5, pp. 1009–1044, 2008.

[8] W. R. Evans and W. D. Davis, "High-performance work systems and organizational performance: The mediating role of internal social structure," *Journal of Management*, vol. 31, no. 5, pp. 758–775, 2005.

[9] C. E. Helfat and M. A. Peteraf, "The dynamic resource-based view: capability lifecycles," *Strategic Management Journal*, vol. 24, no. 10, pp. 997–1010, 2003.

[10] C. A. Hidalgo and R. Hausmann, "The building blocks of economic complexity," *Proceedings of the National Academy of Sciences*, vol. 106, no. 26, pp. 10 570–10 575, Jun. 2009.

[11] W. van der Aalst, *Process Mining: Data Science in Action*, 2nd ed. Springer-Verlag Berlin Heidelberg, 2016.

[12] B. Balassa and M. Noland, ""Revealed" Comparative Advantage in Japan and the United States," *Journal of Economic Integration*, vol. 4, no. 2, pp. 8–15, Sep. 1989.

[13] R. Pastor-Satorras, A. Vázquez, and A. Vespignani, "Dynamical and correlation properties of the internet," *Phys. Rev. Lett.*, vol. 87, p. 258701, Nov 2001.

[14] R. Hausmann and C. A. Hidalgo, "The network structure of economic output," *Journal of Economic Growth*, vol. 16, no. 4, pp. 309–342, Dec 2011.

[15] R. Hausmann, C. A. Hidalgo, S. Bustos, M. Coscia, S. Chung, J. Jimenez, A. Simoes, and M. A. Yildirim, *The Atlas of economic complexity: mapping paths to prosperity*. Cambridge, Mass.: Center for International Development, Harvard University, 2011, oCLC: 781855824.

[16] Van Dongen, B.F. (Boudewijn), "BPI challenge 2015," 2015. [Online]. Available: https://data.4tu.nl/repository/uuid:31a308ef-c844-48da-948c-305d167a0ec1

[17] P. Delias, "A positive deviance approach to eliminate wastes in business processes: The case of a public organization," *Industrial Management & Data Systems*, vol. 117, no. 7, pp. 1323–1339, 2017.

[18] G. Janssenswillen, *bupaR: Business Process Analysis in R*, 2018, r package version 0.4.1.

[19] U. van der Ham, "Benchmarking of five dutch municipalities with process mining techniques reveals opportunities for improvement," Innsbruck, Austria, Sep. 2015. [Online]. Available: https://www.win.tue.nl/bpi/lib/exe/fetch.php?media=2015:bpic2015_paper_11.pdf

# Compliance Monitoring on Process Event Streams from Multiple Sources

Patrik Koenig
University of Vienna
Faculty of Computer Science
Vienna, Austria
Email: patrik.koenig@univie.ac.at

Juergen Mangler
Austrian Center of Digital Production
Vienna, Austria
Email: juergen.mangler@acdp.at

Stefanie Rinderle-Ma
University of Vienna
Faculty of Computer Science, ds:UniVie
Vienna, Austria
Email: stefanie.rinderle-ma@univie.ac.at

***Abstract*—Comprehensive and continuous compliance monitoring is crucial for many process scenarios, e.g., machine-spanning production processes. However, business process execution data is often scattered over several heterogeneous event sources and needs to be seamlessly incorporated into the data structure of the compliance checking system. The proposed COMS approach offers the possibility to define business rules based on process behaviour and the ability to handle events from different information system sources in an integrated way. COMS consists of a generic event data structure to store process information, a rule language, and matching concepts. COMS concepts are prototypically implemented and evaluated based on a real-world event stream from the manufacturing domain.**

## I. Introduction

Compliance monitoring is a major challenge for business processes, particularly if the execution information is distributed over heterogeneous event sources [1], [2], for example, in distributed supply chain scenarios [3]. It requires to ensure and verify business rules over several event streams emitted at runtime. Most compliance monitoring approaches define a compliance language, but abstract from the data structure in which the process information is provided by the event stream [1]. This potentially causes uncertainty in matching process and rule information caused by e.g., heterogeneous labels or formats in [4]. Hence, an accessible integration format is crucial to overcome uncertainty.

Figure 1 displays a loan application process which is distributed across multiple systems. In this case a bank clerk receives a loan application and therefore triggers the process by creating a loan request. This process is then enacted on process engine A, where the first process activity requires the bank clerk to enter the customer data of the loan requester into the system. Even though the input process is pervasive for the bank clerk, the input is entered and processed on a worklist component. Hereby the contextual information is sent from process engine A to the worklist by invoking its service. After the information is correctly entered it is returned to process engine A. The same applies to the following solvency check activity which triggers the creation of a process instance on process engine B. In process engine B the solvency level is either retrieved from the database, if current information is available or from external sources.

Every step of the process execution generates multiple events which stem from multiple information systems with different characteristics. To monitor the process execution all of these events may need to be taken into account to assemble a holistic process context. Moreover, by now process information as represented in the event stream of Fig. 1 needs to be matched to a predefined data schema. This schema is mostly either too specialised for a specific process engine or too generic in a way that it holds only basic attributes of a process instance. In [5], for example, a relational data schema is specified, which considers a defined set of process attributes and characteristics. Every bit of information which cannot be matched to the data schema is omitted. While XES [6] is the de-facto standard input file-format for process mining tools, it is not well-suited for partial event streams from multiple sources, which have to be processed at run-time. While many XES concepts are universal, a generic event data structure (i.e. not a file format) optimized for allowing for efficient access to structured data sent and received by tasks is crucial.

Consider the following business rule imposed on the loan example in Fig. 1: *if the loan request is greater or equal to one million, the solvency level of the customer needs to be at least A, a manager needs to process the request, and the solvency information must not be older than two days.* In the case of Fig. 1 the information necessary to check this rule is distributed across multiple systems and hence requires the provision of an integrated and accessible information based on a generic event data structure. In modern compliance management system, rules are separated from the process model to foster re-usability and manageability of business rules. Hereby the rule concept needs to be matched to the process model. This matching has to be system independent to support the integration of different process aware information system. When matching rules and process models the event and activity labels are probably not a strong identifier. Often other aspects of an activity, e.g., the employed resources, can be more relevant. Based on the problem statement three research questions have been defined:

RQ1 What are the common properties of an event which is produced during process execution? Do events and their information adhere to a certain structure which is present

978-1-7281-0919-0/19/$31.00 ©2019 IEEE
DOI 10.1109/ICPM.2019.00026

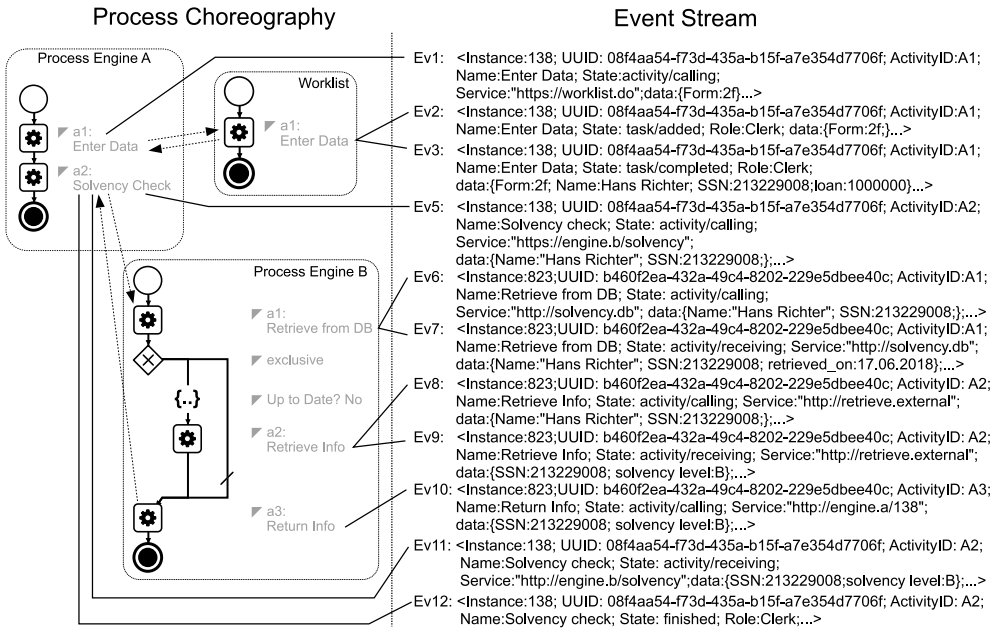



Fig. 1. Loan Process Event Stream Emitted by Two Process Engines

in every event independent of a process engine?

RQ2 Which data structure allows for the most flexible representation of individual events from distributed systems? How can information from enacted business processes be represented efficiently within a runtime compliance monitoring system? Efficient means that the data should be retrievable without traversing the data structure.

RQ3 What language concept enables the simplest but flexible matching of business rules and process events? What is an efficient and flexible way to define such an matching?

For tackling these questions, we develop a Compliance Monitoring System (COMS). It consists, at first, of an accessible data structure to integrate process event streams from multiple information systems (cf. Sect. II) that avoids transformations as much as possible. This comes in hand with a concept for matching business rules and process models beyond label matching that exploits the data exchanged between the different event streams (cf. Sect. III) . The matching goes beyond label matching; instead it matches process rules to process instances based on semantics, e.g., based on the facts which service is invoked by a single process instance activity and the context of the invocation. The proposed artefacts are technically evaluated based on the COMS prototypical implementation and applied to a real-world data set from the manufacturing domain in Sect. IV. Related work is discussed in Sect. V and the paper concludes in Sect. VI.

## II. Generic Event Data Structure

In order to define a generic event data structure for rule and process matching, first it needs to be determined how the information of a process instance is produced and collected. For a process engine, an event is typically produced when the state of a process instance changes, for example, from state "started" to state "finished". Though different granularities of possible states exist, the activities in every process engine follow at least a basic instance activity life-cycle model. This model particularly depends on how the underlying process engine processes single activities. The model of possible states is denoted as *instance activity life-cycle model.*

An instance activity life-cycle model describes the transactions and communication which need to take place to execute a single activity, e.g., a start event, assigning resources to an activity, or the end event. In the following common properties of 4 common life cycle models will be analysed and subsequently transferred into an accessible data model.

**MXML** [7] defines a process log concept. It includes an instance activity life-cycle concept, that does not define process execution states explicitly, but defines the transitions between them. The **Business Process Analytics Format (BPAF)** [8] focuses on the instance activity life-cycle and defines states rather than transitions. Additionally the BPAF model offers a higher granularity of different possible activity states compared to MXML. It also considers resource allocation as states in the instance activity life-cycle model, as well as seven possible end-states, of which MXML has two. Similar to MXML, **XES (eXtensible Event Stream)** [6] describes a language for process logging and aims to solve the problems encountered in MXML. XES references the BPAF life-cycle model as a possible model to describe an instance activity life-cycle, but introduces its own instance activity life-cycle

transitions model. It is similar to BPAF, but also describes the transitions between states. These transitions are similar to those in the MXML life-cycle model. The **CPEE** [9] life-cycle model is a compound instance activity life-cycle model which represents the instance activity life-cycle of the CPEE process engine, including the life-cycle model of a worklist component. The latter is a separate logical system and handles the resource allocation of user based tasks. This makes the system less monolithic as multiple worklists can be connected to the process engine.

Besides the instance activity life-cycle, which is required to correctly address the context of a single events, attributes which need to be present in every event are a unique instance identifier and an identifier for the process activity the event relates to. This is required to correlate all events to a specific process instance and activity. Those two classifiers need to be included into every event otherwise an exact correlation of related events is hardly possible for complex instance activity life-cycle models.

In summary, a single event can be uniquely distinguished based on the following characteristics:

- **A unique instance identifier.** Based on this identifier the process instance can be uniquely identified.
- **An identifier for the activity.** A field or value which identifies a activity within a specific instance.
- **The source system.** As multiple information systems can be involved, this identifies the system the event stems from.
- **The event topic.** Defines the parent state in which context the event was raised.
- **The event name.** It defines the actual event such as calling, assigned, etc. and has to be unique in combination with the event topic within the instance activity life-cycle.

The definition of a source systems provides the possibility to correlate the events of multiple information systems into one process instance trace. Those information systems can either share the same instance activity life-cycle or operate on a different one. This leads to the hierarchic data structure shown in Fig. 2. Every hierarchy level is fully dependent on the combination of all hierarchy levels above. To represent this hierarchical structure, the data structure is defined as interleaved key-value pairs. Finally in the lowest level of the data structure, the actual **event data** is stored as arbitrary semi-structured information, i.e., no structural restriction to the event itself arises.

The example stream in Lst. 1 shows parts of an event stream based on the example in Fig. 1. Lst. 1 also depicts that the actual data is saved in the leaf called “Event Data”, as defined in Fig. 2. The other keys are primarily pointing to the next key which narrows down the specific classification of the event.

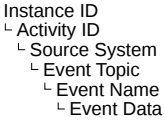



Fig. 2. Data Structure

Listing 1. Event Stream Represented in Generic Event Data Structure

```
:08f4aa54−f73d−435a−b15f−a7e354d7706f:
  :A1:
    :engine:
      :source: Process Engine A
      :activity:
        :calling:
          UUID: 08f4aa54−f73d−435a−b15f−a7e354d7706f
          label: Enter Data
          endpoint: https://worklist.do
          parameters:
            arguments:
              form: http://worklist.do/form/2f.html
              role: Clerk
              user: Hans Richter
        :receiving:
          label: Enter Data
          endpoint: https://worklist.do
          received:
            − Form: http://worklist.do/form/2f.html
            − Name: Hans Richter
            − SSN: 213229008
            − loan: 1000000
    :Worklist:
      :task:
        :completed:
          UUID: 08f4aa54−f73d−435a−b15f−a7e354d7706f
          activityName: Enter Data
          parameters:
            Form: http://worklist.do/form/2f.html
            Name: Hans Richter
            SSN: 213229008
            loan: 1000000
:b460f2ea−432a−49c4−8202−229e5dbee40c:
  :A1:
    :engine:
      :source: Process Engine B
      :activity:
        :calling:
          UUID: b460f2ea−432a−49c4−8202−229e5dbee40c
          label: Retrieve from DB
          endpoint: http://solvency.db
          parUUID: 08f4aa54−f73d−435a−b15f−a7e354d7706f
          parameters:
            arguments:
              Name: Hans Richter
              SSN: 213229008
        :receiving:
          UUID: b460f2ea−432a−49c4−8202−229e5dbee40c
          label: Retrieve from DB
          endpoint: http://solvency.db
          parUUID: 08f4aa54−f73d−435a−b15f−a7e354d7706f
          received:
            − Name: Hans Richter
            − SSN: 213229008
            − retrieved_on: ’2018−03−17T11:21:37.213+02:00’
```

## III. Matching Business Rules and Processes

Section III-A starts with a discussion of different approaches for rule and process matching. The rule schema used for flexible matching to processes is described in Sect. III-B. Section III-C finishes with the checking mechanisms.

### A. What to match?

A first idea could be to base rule and process matching on unique IDs as every process activity has a unique id based on which it can be distinguished. The mayor drawback in this case is that this unique identifier is hard to maintain within the business rule as well as the process repository. Moreover, in case of process evolution it is hard to determine if the rule still applies since a unique id neither represents the context of an activity nor its execution.

Process matching, mining, and compliance are widely based on label equivalence [10], i.e., two process activities or an event and a process activity are considered as equal if their

labels are equivalent. More flexibility was added by going from label equivalence to label similarity based on string similarity metrics such as string edit distance [11]. However, label equivalence and similarity are not useful in the context of multiple and heterogeneous data sources. Hence, in literature it is proposed to shift from label equivalence to attribute equivalence [10], i.e., qualifying a set of attributes of a process activity or event that signifies the equivalence and hence the matching. This is suitable for runtime evaluation as the context of the activity can be determined even though the full execution trace is not available. Examples for attributes comprise resources, service endpoints, and service parameters. In connection with service endpoints it has to be noted that different web services URIs can offer identical web services. Service parameters are sent to a web service during the service call. As discussed in [10], these service parameters provide attributes which describe the context of a service, e.g., the use of a specific form in a specific web service.

The COMS matching approach works with attribute equivalence. For this, in the data structure defined in Fig. 2 every characteristics of an activity, which is included in the events, is represented as an attribute. Due to the hierarchical structure, instead of referring to a fixed attribute set, different attribute levels can be considered for matching. More precisely, to match the process rules to process activities a single identifier or a combination of the above can be used. Moreover, the matching semantics can be defined specifically for each rule as described in the next section, i.e., based on a novel Match-Condition-Action rule structure.

### B. Rule Schema

In order to incorporate the flexible definition of the rule matching into the rule the well-known principle of Event-Condition-Action rules as, for example, used in previous work [12] is adapted to a **Match-Condition-Action** rule. Here the **Match** part defines how events are matched to process activities contained in the rule, i.e., based to which attribute set a matching can be successfully conducted. Consider the rule (notation: YAML) as shown in Lst. 2. The **Match** part of the rule defines on which business process activity the rule applies. Multiple activities can be defined and then are internally associated with a symbol, which is later used to describe conditions which relate to multiple activities. Such symbols are represented in Rows 2, 6, and 12 in Lst. 2. The next section of the rule schema contains the conditions. They are checked if an activity pattern (e.g., "A precedes B" [1]) matches the conditions in the match section of the rule. If a match pattern is sufficiently fulfilled, a statement concerning the compliance of the instance can be made. In the match part as well as in the conditions part, logical operations can be used to describe the connection between the different clauses. If the connection is not explicitly defined, an "AND" connection is assumed. A clause can span across multiple rows, but also can be defined within one row. Depending on how the conditions evaluate, the actions defined in the *"if"* or in the *"ifnot"* part are executed. The *"if"* part applies if the outcome of the evaluation of the conditions is true. The *"ifnot"* part is applied if the conditions evaluate as false. The actions are enacted as soon as a definitive outcome can been determined, this can be the case even before every condition is evaluated. For example if all conditions are conjunct, as soon as one evaluates to FALSE, the outcome of the compliance rule is FALSE.

TABLE I
CLAUSE STRUCTURE

| **left hand side** | **operator** | **right hand side** |
|---|---|---|
| system > topic > name > data | == | "http://example.org/service" |
| system > topic > name > data | <= | 300 |
| system > topic > name > time | $before$ | system > topic > name > time |

Listing 2. Example Compliance Rule

```
1 match:
2  a :
3   − ["engine > activity > calling > endpoint","==","https://worklist.do"]
4   − ["engine > activity > calling > parameters > arguments > form",
5      "==","http://worklist.do/form/2f.html"]
6  b :
7   − ["engine > activity > calling > endpoint","==","http://engine.b"]
8   − ["engine > activity > calling > parameters > arguments",
9      "include?","SSN"],
10  − ["engine > activity > calling > UUID",
11     "==", "engine > activity > calling > parUUID"
12 c:
13  − ["engine > activity > calling > endpoint",
14     "==","http://solvency.db"]
15  − ["engine > source","==","Process Engine B"],
16  − ["engine > activity > calling > UUID",
17     "==", "engine > activity > calling > parUUID"
18 condition:
19  − ["a > engine > activity > receiving > received > loan","<=",1000000]
20  − "IMPLIES"
21  − "("
22   − ["a > worklist > user > take > Role","==","Manager"]
23   − ["b > engine > activity > receiving > received > solvency_level",
24      "==","A"]
25   − ["c > engine > activity > calling > parameters > retrieved_on",
26      "withindays",2]
27  − ")"
28 if: []
29 ifnot: [notify_warn]
```

*1) Rule Clauses.:* The clauses express compliance conditions. As seen in Lst. 2 clauses are defined in both the Match and Condition section, as for example in Rows 3 or 17. In accordance with Compliance Monitoring Functionality 2 on "data" as set out in [1] we support *unary* and *extended* conditions of the form "$d \odot v$" where d is a process data element and $\odot$ is a comparison operator. For unary conditions, "$v$ is some value of $d$'s domain" [1]. Examples are Clauses 1 and 2 in Tab. I. For extended conditions $v$ refers to paths which reference values within the data structure. Then basically two values are processed against each other whereby one of them constitutes a reference to the data structure. An example is Clause 3 in Tab. I.

*2) Referencing data structure values.:* Based on the underlying instance activity life-cycle model, COMS monitors specific events which are defined in the rule clause. These references are therefore paths which seek for the relevant event data. The paths follow the same structure as shown in Fig 2 except for the first two hierarchy levels. They are abstracted in the reference, as they identify the process activity instance within the data structure.

TABLE II
RULE REFERENCE STRUCTURE – EXAMPLES

| source | event topic | event name | event data structure | | |
|---|---|---|---|---|---|
| engine | activity | calling | endpoint | | |
| engine | activity | calling | parameters | arguments | form |
| engine | dataelements | change | changed | schadenssumme | |
| worklist | user | take | user | | |
| worklist | user | giveback | cpee_activity | | |

*a) Path.:* The basic syntax for reference data values, described by a regular expression, is as follows: `/^(\w|\s|\>)+\w+\$/`

A sequence of characters is defined by a word as a start, followed by a ">", which operates as separator. This pattern can be repeated multiple times and finally the query must end with a word. Every element between the ">" character is hereby an entity which is looked up. The first entity hereby defines the element with the highest hierarchy, going deeper with each given entity. Compared to the first three hierarchy levels as displayed in Fig. 2, the level "Event Data" can have an arbitrary depth. An example for this rule reference structure is represented in Tab. II: the hierarchy elements in the three leftmost columns relate to the event. The elements in the two rightmost columns relate to the event data attributes.

### C. Rule Checking

If the process matches the semantics defined in the match section of a rule, the activity is mapped to a symbol. For the rule example in Lst. 2, the symbols "a", "b" and "c" are later used in the conditions section, to define which condition should be applied to which activity.

In order to express structural compliance conditions, currently, COMS supports selected LTL patterns [13], i.e., (eventual) existence of an activity, absence of an activity, and a follows relations between activities. We have evaluated the expressiveness of COMS rules along the "Compliance Monitoring Functionalities framework (CMFF)" defined in [1]. COMS exhibits full coverage of CMF1 – 3 regarding data, time, and resources. Due to the explicit support of the instance activity life cycle, the execution CMF 4 – 6 are well-supported. Regarding the user-oriented CMF 7 – 10 COMS particularly supports root cause analysis as the source (system) of compliance violations can be explicitly determined.

*1) Conditions.:* Rule queries within the conditions must also include a reference to the matched activities. This reference is represented in the first entity of the rule query. For example the rule query "$a > worklist > user > take$" within Lst. 2 would reference the activity "$a$" defined in the "*match*" part. Therefore the value will be retrieved from the activity which matched the defined pattern in "$a$". A rule file can include multiple conditions. Conditions are, by default joined with an "AND" operator. Additionally they can be connected with an "OR" or an "IMPLIES" operator.

*2) DSL.:* To be able to define more complex conditions, in the operator of a clause, a DSL is introduced. Besides the basic operators "==", "<=", "!=", additional operators can be used. These operators depend on the data type and allow all operators for this type based on the ruby programming language as well as the possibility to define custom operators.

*3) Actions.:* Depending on the outcome of the conditions actions are executed. In case the overall condition evaluates as true the "if" part is executed, otherwise the "ifnot" part will be invoked. In the example rule in Lst. 2, the actions are defined in Rows 28 and 29.

Actions are code which is supplied with the rules to the compliance management system. It uses a blackboard concept [14] to share data. As actions are executed they may persist data on the blackboard (e.g. store an average of certain values). Blackboard data may either be shared between all event sources, or between all tasks/event sources inside an instance (cmp. to Lst. 1):

Listing 3. Example Data represented in YAML

```
1 :08f4aa54−f73d−435a−b15f−a7e354d7706f:
2   :Enter Data: ...
3   :Retrieve from DB: ...
4   :Blackboard:
5     event: changed
6     values:
7       somevalue: 1
8 :Blackboard: (see above)
```

Changing a value on a blackboard generates an additional event, that can be caught by the compliance management system. Thus, as seen in Lst. 3, the blackboard is treated just like any event source, being either equal to an activity (e.g. "Enter Data") or an instance (e.g. :08f4aa54...).

### D. Matching Performance

When looking at the way the data and the events are structured, and the examples above, it becomes apparent that the complexity of the rule-base and the events is fairly limited (see also CRISP rule-base [15]). Thus we can optimise the algorithm that compares the rules with the events.

To evaluate the defined rules, during initialisation all rules are parsed and common clauses are joined into one matching set. For example all rules with the clause ["engine > activity > calling > endpoint","==","https://worklist.do"] will be joined, as shown in Lst. 2. The resulting list of clauses point to each rule the are contained in, and how many clauses each originating rule contains.

Listing 4. Joining rules to matching sets

```
1 def join_rules(rulesets)
2   matchbase = new Hash
3   foreach rule in ruleset
4     foreach clause in rule
5       matchbase[clause] << [rule_id,number_of_clauses]
6     end
7   end
8 end
```

It is important to note, that there is no differentiation between clauses in events and clauses in condition necessary, as they all have to match in order for an action to be triggered.

The resulting match-base is a hash that can be accesses with O(1) if the key is known. In an incoming event, which is used to prepare a blackboard view, each value is a leaf, and the path to each leaf is the key that can be used to access the the condition and information about matching rules in the ruleset.

Listing 5. Store matches of rules

```
def on_event(event)
  foreach leaf in event
    if clause = matchbase[leaf] then
      foreach ref in clause
        matches[instance + task][ref->rule_id] << leaf
        if count(
          unique(
            matches[instance + task][ref->rule_id]
          )
        ) == ref->number_of_clauses
          ACTION
        end
      end
    end
  end
end
```

As can be seen above, access to the match-base is constant, the runtime of the algorithm is mostly dependent on the pieces of information in each event.

## IV. Evaluation

The *feasibility* of COMS is demonstrated based on a prototypical implementation, which is made available here[1], also including the logs used for the evaluation.

### A. Evaluation of Applicability

The approach is also evaluated with respect to its *applicability* based on a real world event stream of a manufacturing process operated at the Austrian Competence Center for Digital Production (CDP)[2]. We analyze the event stream emitted by a EMCO "MaxxMill 500" during the execution of a process instance which orchestrates the milling of a part. After triggering the manufacturing of the part, the process instance running on the Cloud Process Execution Engine (CPEE) [9] collects data from the mill from multiple sources/sensors simultaneously. Through one sensor connected by the MT-Connect[3] [16] protocol it is possible to detect the overall power consumption of the machine. The goal is, based on energy consumption, to find out about (1) explicit machine errors during milling and (2) to monitor the power level during milling. While the mill has some explicit error states, it cannot detect (a) when a milling tool breaks and (b) when the edge of a milling tool rapidly deteriorates.

For the approach presented it this paper, it makes no difference if the event stream origins from one source, or many, as it is collected into one generic **event data** structure.

We created a set of rules (including custom actions) that detect Errors (a) and (b) from the power levels during milling. If the milling tool breaks (and falls off) the power levels show a rapid decline, as there is no more resistance. If the milling tools' edge becomes suddenly blunt the power levels spike. For both cases we want the process (and thus the milling) to stop, so that the machine can be fixed. This minimises the time wasted due to faulty milling tools. The realisation of this scenario is based on events such as depicted in Lst. 6. The unit for the value (power consumption) is watt.

[1]https://github.com/pakoe/coms

[2]http://www.acdp.at/

[3]http://www.mtconnect.org/

Listing 6. Aggregated Event Split Into Single Events

```
event:
  cpee:lifecycle:transition: activity/receiving
  list:
    data_receiver:
      message:
        mimetype: application/json
        content:
          ID: pac51_65
          value: 1956.174
```

The realization was achieved by two simple rules. The rule depicted in Lst. 7 monitors the operation of the milling machine. In Rows 3 to 4 of Lst. 7, the milling task "MaxxMill 500" is identified based on the service invoked by the process instance. All activities, in all instances which make use of the "MaxxMill 500" are monitored.

Listing 7. Sensor Data Rule 1

```
match:
  - :a :
    - [ "engine > activity > done > endpoint", "==", "https://centurio.work/data
        /mm500/signals/"],
    - [ "engine > source", "==", "MaxxMill 500"]
condition: [ [ "a > engine > activity > receiving > received > message >
      content > pac52_65 > value", "exits?" ] ]
if: [blackboardize_power_value]
ifnot: []
```

The condition evaluates if the power value is present in the respective activity. If yes the value is saved to a blackboard as described in Sec. III-C3 (see Lst. 8), which is accessible across all process instances.

Listing 8. Action blackboardize_power_value

```
def blackboardize_power_value(ctx)
  value = ctx.a.engine.activity.receiving.received.message.content.pac52_65.value
  ctx.blackboard.values.last_value = value
end
```

The change of a value within the blackboard then triggers the rule depicted in Lst. 9.

Listing 9. Sensor Data Rule 2

```
match:
  - :a : [ [ "blackboard > event", "==", "changed" ] ]
condition:
  - [ "a > blackboard > values > last_value", "<", "a > blackboard > values
      > average_low" ]
  - [ "a > blackboard > values > last_value", ">", "a > blackboard > values
      > average_high" ]
if: [engine_stop]
ifnot: [calculate_average]
```

Its condition evaluates if the latest sensor is out a certain power consumption band (lower bound, upper bound). If the value is outside of the band, the machine is stopped. If the value is within the defined band, the value contributes to the average power consumption (see action depicted in Lst. 10).

Listing 10. Action calculate_average

```
def calculate_average(ctx)
  ctx.blackboard.values.average =
    ( ctx.blackboard.values.average * ctx.blackboard.values.average_count +
      ctx.blackboard.values.last_value) /
    ( ctx.blackboard.values.average_count + 1 )
  ctx.blackboard.values.average_count += 1
  ctx.blackboard.values.average_low = ctx.blackboard.values.average - 10
  ctx.blackboard.values.average_high = ctx.blackboard.values.average + 10
end
```

There are various benefits to this approach. As mentioned above, errors can be detected and fixed early, which leads to

higher Overall Equipment Effectiveness (OEE) [17]. While this could be done through explicit process modelling, this would require explicit tasks for sharing the average values, which is not part of the business logic and complicates the process model. Thus a specific advantage of our approach is, that it is automatically affecting all current and future process models which invoke the milling machine.

### B. Evaluation of Computational Complexity

As elaborated above, based on the example rule-set we make the following assumptions.

- Rule-sets are large, individual events are small.
- Actions are not allowed to trigger rules.

We then designed the data-structures around these decisions to allow for an algorithm that can yield optimal performance for the given problem. In comparison to more generic algorithms like RETE, which support deeply intertwined rule-bases, this allows for better runtime characteristics. In Tab. III, we outline some popular solutions, which could achieve the same results as our solution, and their runtime characteristics.

In the RETE Algorithm [18], the matchset is denoted as productions $C$ (see Lst. 2), while the event data (see Lst. 1) is denoted as working memory $W$. Each individual line, in each event is thus a working memory entry *WME*. In the simplest case, whenever a new *WME* occurs, the runtime is $C$. However in RETE, things might get worse, as for $C$ it is assumed that all elements of the matchset are strictly independent, basically forming one rule. If they are not, then $P = NP$.

Our algorithm is designed with the following properties (speaking in RETE terms):

- Each *WME* should be compared to $C$ only once.
- Each element in $C$ is independent, thus the problem is not NP hard, no trigger cascades can occur.
- Matching $C$ is O(1) as we can utilise a hash structure, and do not have to deal with true conditions in the sense of RETE.

The problem we tackled, could be solved in RETE, but the runtime would be $W^C$, whereas the runtime for the algorithms shown in Lst. 5 is in the worst case $W * C$ (i.e. multilinear). As concluded by [19] in Chapter 6.1, we assume restrictions on $W$ and $C$, which will lead to polynomial runtime in $W$ and $C$, while the RETE algorithm is able to tackle much more complex problems, which can be linear in $P$.

A comparison of the algorithms shown Lst. 5 to techniques utilised in Prolog, seems fairer. Tabling should yield similar results, to our approach, whereas standard search and backtrack should be similar to the results given by RETE networks.

In Tab. III a summary of this discussion is shown, including the names of popular RETE based engines. For the case of Drools we assume that the Lazy RETE algorithm (asserting possible solutions in the network) can lead to significantly improved runtime, although its remains unclear if it can come close to $W * C$, as this step adds additional in relation to $C$

So a good solution in Prolog should perform as well as any implementation that relies on the algorithm shown in Lst. 5. As mentioned before, our prototypical implementation is a micro-service with a REST interface relying on the Cloud Process Execution Engine for the data-streams, which is tedious in Prolog, but easy in any modern programming language.

TABLE III
RUNTIME CHARACTERISTICS

| Prolog | | Drools | | Jessy | COMS |
|---|---|---|---|---|---|
| Tabling | Search & Backtrack | Rete | Lazy Rete | Rete | Hash Access |
| $W * C$ | $W^C$ | $W^C$ | $< W^C$ | $W^C$ | $W * C$ |

## V. RELATED WORK

The compliance of process models can be monitored online based on on event streams [1]. Online conformance checking [20] measures the conformance of process models with some behaviour of interest (e.g., expressed by behavioural patterns) based on events streams during runtime. COMS concepts can be used to integrate event streams for compliance monitoring and online conformance checking. As stated in [21] *"[s]tate-of-the-art conformance analysis techniques are typically optimised and devised for one-time use"*. While [21] advocates performance gains by conformance approximation, COMS designs its rules less expressive as declarative approaches such as MobuconLTL and MobuconEC (cf. pattern-based analysis in [1]), but "expressive enough" to be rapidly checked.

[22] proposes techniques "for aligning business process compliance and monitoring requirements in dynamic [business networks]". However, the focus is not on the process and rule matching at the data level. Choreography compliance has been analysed in [23], [3]; the proposed compliability criterion refers to the property that a collaboration can comply with global and local compliance rules as well as assertions. Again the data aspect has not been considered here. Event-based compliance checking in business processes comprises approaches for Complex Event Processing (CEP). In [24] the events in the stream can be aggregated to match activities in the process model. [25] works in a similar way for event stream integration for manufacturing processes. Both approaches provide valuable mechanisms for aligning the process and event stream level, however, do not directly address uncertainty in the context of activity labels and different formats. The same holds for previous work on matching rules and processes [26], [12]. Several approaches address the matching between process models, e.g., [27] and process similarity (see survey in [11]). Here similarity between process models is based on label equivalence or similarity measures such as string edit distance. As suggested in [10] the presented approach abstracts from label comparison and uses attribute equivalence instead. The observation that a process activity consist out of more than one event and therefore the consideration of an instance activity lifecyle is also present in [28]. [28] applies a m:n mapping of events to predefined business processes based on the activity labels and textual description. Our work in contrast (1) operates at runtime and (2) utilises a multitude of aspects, while in the context of this paper relying on non-fuzzy properties of process execution, such as task IDs,

endpoints and exchanged data.

COMS can be also compared to existing business rule engines such as Drools [29]. In contrast to, e.g., Drools, our approach models the process execution domain itself, with multiple instances, activities, instance activity life-cycle stages and event attributes. Therefore the domain does not have to be modelled to save the data or make the information processable. The structure of the rule clauses is to some point similar to XPATH [30], this similarities arise as they both operate on tree structured data. COMS processes the information not as a whole, but incrementally, with every incoming event.

## VI. Conclusion

The presented COMS framework supports flexible matching of business rules and process events from multiple sources. This, in turn, enables integrated compliance checking together with resolving uncertainty caused by heterogeneous labelling and different formats. COMS consists of a generic event data structure for storing events, a generic rule language, and concepts for matching process tasks and events. The evaluation shows that it is applicable to real-world scenarios and the concept is mature enough to be adapted to newly emerging requirements. COMS creates an holistic approach, by defining a fitting data structure for the process event streams and offers a rule language which is designed to operate on this structure. Future work will develop a graphical user interface and investigate on how to involve users into process and rule matching most efficiently. Moreover, we aim at conducting further case studies, e.g., in the logistics domain.

**Acknowledgements:** This work has been partly funded by the Vienna Science and Technology Fund (WWTF) through project ICT15-072 and the Austrian Research Promotion Agency (FFG) via the "Austrian Competence Center for Digital Production" (CDP) under the contract number 854187.

## References

[1] L. T. Ly, F. M. Maggi, M. Montali, S. Rinderle-Ma, and W. M. P. van der Aalst, "Compliance monitoring in business processes: Functionalities, application, and tool-support," *Information Systems*, vol. 54, pp. 209 – 234, 2015.

[2] M. Wang, K. Y. Bandara, and C. Pahl, "Distributed aspect-oriented service composition for business compliance governance with public service processes," in *Internet and Web Applications and Services*, 2010, pp. 339–344.

[3] W. Fdhila, S. Rinderle-Ma, D. Knuplesch, and M. Reichert, "Change and compliance in collaborative processes," in *Services Computing*, 2015, pp. 162–169.

[4] A. Artikis, A. Gal, V. Kalogeraki, and M. Weidlich, "Event recognition challenges and techniques: Guest editors' introduction," *ACM Trans. Internet Techn.*, vol. 14, no. 1, pp. 1:1–1:9, 2014.

[5] J. Schiefer, B. List, and R. M. Bruckner, "Process data store: A real-time data store for monitoring business processes," in *Database and Expert Systems Applications*, 2003, pp. 760–770.

[6] IEEE, "Ieee standard for extensible event stream (xes) for achieving interoperability in event logs and event streams," *IEEE Std 1849-2016*, pp. 1–50, Nov 2016.

[7] B. F. van Dongen and W. M. Van der Aalst, "A meta model for process mining data." *EMOI-INTEROP*, vol. 160, p. 30, 2005.

[8] M. zur Muehlen and K. D. Swenson, "BPAF: A standard for the interchange of process analytics data," in *Business Process Management Workshops*, 2010, pp. 170–181.

[9] J. Mangler, G. Stuermer, and E. Schikuta, "Cloud process execution engine - evaluation of the core concepts," CoRR, Tech. Rep., 2010.

[10] S. Rinderle-Ma, M. Reichert, and M. Jurisch, "On utilizing web service equivalence for supporting the composition life cycle," *Int. J. Web Service Res.*, vol. 8, no. 1, pp. 41–67, 2011.

[11] R. M. Dijkman, M. Dumas, B. F. van Dongen, R. Käärik, and J. Mendling, "Similarity of business process models: Metrics and evaluation," *Inf. Syst.*, vol. 36, no. 2, pp. 498–516, 2011.

[12] J. Mangler and S. Rinderle-Ma, "Rule-based synchronization of process activities," in *Commerce and Enterprise Computing*, 2011, pp. 121–128.

[13] F. M. Maggi, M. Westergaard, M. Montali, and W. M. P. van der Aalst, "Runtime verification of ltl-based declarative process models," in *Runtime Verification*, 2011, pp. 131–146.

[14] D. D. Corkill, "Blackboard systems," *AI expert*, vol. 6, no. 9, pp. 40–47, 1991.

[15] W. Fdhila, M. Gall, S. Rinderle-Ma, J. Mangler, and C. Indiono, "Classification and formalization of instance-spanning constraints in process-driven applications," in *International Conference on Business Process Management*. Springer, 2016, pp. 348–364.

[16] A. Vijayaraghavan, W. Sobel, A. Fox, D. Dornfeld, and P. Warndorf, "Improving machine tool interoperability using standardized interface protocols: Mt connect," in *Flexible Automation*, 2008.

[17] Ö. Ljungberg, "Measurement of overall equipment effectiveness as a basis for tpm activities," *International Journal of Operations & Production Management*, vol. 18, no. 5, pp. 495–507, 1998.

[18] C. L. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem," in *Readings in Artificial Intelligence and Databases*, J. Mylopolous and M. Brodie, Eds. San Francisco (CA): Morgan Kaufmann, 1989, pp. 547 – 559.

[19] R. B. Doorenbos, "Production matching for large learning systems." Carnegie-Mellon Univ., Dept. of Computer Science, Tech. Rep., 1995.

[20] A. Burattin, S. J. van Zelst, A. Armas-Cervantes, B. F. van Dongen, and J. Carmona, "Online conformance checking using behavioural patterns," in *Business Process Management*, 2018, pp. 250–267.

[21] P. M. Dixit, H. M. W. Verbeek, and W. M. P. van der Aalst, "Fast conformance analysis based on activity log abstraction," in *Enterprise Distributed Object Computing*, 2018, pp. 135–144.

[22] M. Comuzzi, "Alignment of process compliance and monitoring requirements in dynamic business collaborations," *Enterprise IS*, vol. 11, no. 6, pp. 884–908, 2017.

[23] D. Knuplesch, M. Reichert, W. Fdhila, and S. Rinderle-Ma, "On enabling compliance of cross-organizational business processes," in *Business Process Management*, 2013, pp. 146–154.

[24] E. Mulo, U. Zdun, and S. Dustdar, "Domain-specific language for event-based compliance monitoring in process-driven soas," *Service Oriented Computing and Applications*, vol. 7, no. 1, pp. 59–73, 2013.

[25] S. Appel, P. Kleber, S. Frischbier, T. Freudenreich, and A. P. Buchmann, "Modeling and execution of event stream processing in business processes," *Inf. Syst.*, vol. 46, pp. 140–156, 2014.

[26] C. Indiono, J. Mangler, W. Fdhila, and S. Rinderle-Ma, "Rule-based runtime monitoring of instance-spanning constraints in process-aware information systems," in *On the Move to Meaningful Internet Systems*, 2016, pp. 381–399.

[27] H. Leopold, M. Niepert, M. Weidlich, J. Mendling, R. M. Dijkman, and H. Stuckenschmidt, "Probabilistic optimization of semantic process model matching," in *Business Process Management*, 2012, pp. 319–334.

[28] T. Baier, C. Di Ciccio, J. Mendling, and M. Weske, "Matching events and activities by integrating behavioral aspects and label analysis," *Software & Systems Modeling*, vol. 17, no. 2, pp. 573–598, 2018.

[29] M. Proctor, "Drools: A rule engine for complex event processing," in *Applications of Graph Transformations with Industrial Relevance*, 2012, pp. 2–2.

[30] J. Robie, J. Snelson, D. Chamberlin, and M. Dyck, "XML path language (XPath) 3.0," W3C, W3C Recommendation, Apr. 2014, http://www.w3.org/TR/2014/REC-xpath-30-20140408/.

# Prediction-based Resource Allocation using LSTM and minimum cost and maximum flow algorithm

Gyunam Park and Minseok Song
Department of Industrial and Management Engineering
POSTECH (Pohang University of Science and Technology)
77 Cheongam-Ro. Nam-Gu. Pohang. South Korea 37673
{gnpark, mssong}@postech.ac.kr

***Abstract*—Predictive business process monitoring aims at providing the predictions about running instances by analyzing logs of completed cases of a business process. Recently, a lot of research focuses on increasing productivity and efficiency in a business process by forecasting potential problems during its executions. However, most of the studies lack suggesting concrete actions to improve the process. They leave it up to the subjective judgment of a user. In this paper, we propose a novel method to connect the results from predictive business process monitoring to actual business process improvements. More in detail, we optimize the resource allocation in a non-clairvoyant online environment, where we have limited information required for scheduling, by exploiting the predictions. The proposed method integrates offline prediction model construction that predicts the processing time and the next activity of an ongoing instance using LSTM with online resource allocation that is extended from the minimum cost and maximum flow algorithm. To validate the proposed method, we performed experiments using an artificial event log and a real-life event log from a global financial organization.**

## I. Introduction

Process mining techniques allow the extraction of in-depth insights in process-related problems, which business corporations face, from event logs available in Process-Aware Information Systems(PAISs) [1]. Through the application of process mining, companies can discover the process model describing how they work, examine the difference between a reference process model and a discovered one, and calculate a variety of performance measures such as case duration, resource utilization, and bottlenecks in processes [2]. These techniques are commonly applied in an offline fashion where only completed cases are being considered. Recently, however, online process mining techniques, which considers running cases, are gaining more interests [3].

In process mining, predictive business process monitoring provides the predictions concerning the future status of ongoing cases of a business process [4]. It aims at improving business processes by offering timely information that enables proactive and corrective actions [5]. Previous studies focus on the prediction tasks (e.g., time, risk probability, performance indicators, and next event) using several methods such as annotated transition system, machine learning, or statistics [5], [6], [7].

The previous studies, however, do not suggest how the prediction results can be exploited to improve business processes, leaving it up to the subjective judgment of a user [2]. In order to achieve the goal of process improvement, the prediction results should be transformed into concrete improvement actions. In this regard, we demonstrate how the results from predictive process monitoring can be transformed to suggest an actionable recommendation on resource allocation aiming at improving business processes.

Resource allocation is an essential and challenging problem in business process management (BPM) [8]. Efficient resource allocation improves productivity, balances resource usage, and reduces execution costs [9]. Resource allocation in BPM shares commonalities with the job-shop scheduling [10]. The problem is to find the job sequences on machines to achieve a goal (e.g., minimize makespans), while the machine sequence of the jobs is fixed [11]. There has been considerable research in the area of job shop scheduling over the past years [12]. As it is NP-hard [13] and one of the most computationally intractable combinatorial problems [14], heuristic techniques have been developed such as dispatching rules, shifting bottleneck heuristic, and local search [12]. Among those techniques, dispatching rules receive massive attention from a practical viewpoint [15]. A dispatching rule is used to assign a job to a resource at a given time when a resource becomes available for the operation. This approach is useful to find a reasonably good schedule concerning an objective such as the makespan, the total completion time, or the maximum lateness in a relatively short time.

A dispatching rule is only applicable when we are aware of required parameters such as the release time, the processing time, the sequence of operations of jobs. In many circumstances, however, we have limited information about the scheduling parameters [16]. Imagining an emergency department of a hospital, we do not know when and why a patient would come into the department before the visit happens. Furthermore, there can be irregular clinical procedures even for the patients diagnosed with the same disease since exceptions are always able to occur. Even worse is that we are unaware of the processing time taken to finish an operation, making it difficult to assign the most efficient resource to patients. This problem is called a non-clairvoyant online-over-time problem [17] where the decision-maker do not know the data concerning a job when the job is presented to him. In this problem, we cannot apply the developed dispatching rules to

978-1-7281-0919-0/19/$31.00 ©2019 IEEE
DOI 10.1109/ICPM.2019.00027

make an optimal decision.

A natural idea to deal with the above example is first to predict relevant parameters and then utilize the predictions to optimize the resource scheduling. To achieve this goal, two challenges need to be addressed. (i) How to effectively build prediction models to generate the required parameters? (ii) How to efficiently dispatch resources based on the predictions? Thus, we propose a two-phase method to optimize resource allocation based on prediction to efficiently solve the non-clairvoyant online problem under the objective of minimizing total weighted completion time.

In this paper, we demonstrate how the results from predictive business process monitoring can be transformed into a concrete process improvement action. To this end, we solve a non-clairvoyant online-over-time problem with the two-phase method integrating predictions with resource allocation optimization. To the best of our knowledge, this is the first work to solve the problem by associating results from predictive business process monitoring. To verify the effectiveness and efficiency of our proposed method, we evaluate it on both an artificial and a real-life event log.

In the rest of this paper, we explain backgrounds in Section II and provide a running example, a baseline approach and insights to solve the problem in Section III. In Section IV, we detail the two-phase method, i.e., offline prediction model construction and online resource scheduling. Section V presents how we evaluate our suggested method both on artificial and real-life event logs. Then we review related work in Section VI and concludes this paper in Section VII.

## II. Backgrounds

This section presents preliminaries and notations that will be required in the remainder of this paper. We also elaborate non-clairvoyant online job shop scheduling problem we try to solve in this paper.

### A. Preliminaries

Let $\mathcal{E}$ be the event universe. Each event is characterized by its attributes such as activity, resource, timestamp.

**Definition 1** (Trace, Event Log)**.** *A trace $\sigma = e_1,e_2,...,e_n \in \mathcal{E}^*$ such that each event occurs only once, i.e., for $1 \leq i < j \leq |\sigma| : e_i \neq e_j$. An event log $\mathcal{L}$ is a multi-set of traces such that each event appears at most once in the entire log.*

**Definition 2** (Prefix, Suffix)**.** *A prefix of length $k$ ($0 < k < n$) of a trace $\sigma = \langle e_1, e_2, ..., e_n \rangle$ is $h_k(\sigma) = \langle e_1, e_2, ..., e_k \rangle$ and its suffix is $t_k(\sigma) = \langle e_{k+1}, ..., e_n \rangle$. For instance, for $\sigma_i = \langle e_1, e_2, e_3, e_4, e_5 \rangle$, $h_3(\sigma_i) = \langle e_1, e_2, e_3 \rangle$ and $t_3(\sigma_i) = \langle e_4, e_5 \rangle$.*

Let $\mathcal{A}, \mathcal{R}$ and $\mathcal{T}$ be the set of activities, resources, and processing times, respectively.

**Definition 3** (Event representation function)**.** *A function $\pi_{\mathcal{A}} \in \mathcal{E} \rightarrow \mathcal{A}$ assigns process activities to each event, and a function $\pi_{\mathcal{R}} \in \mathcal{E} \rightarrow \mathcal{R}$ assigns to each event a resource. A function $\pi_{\mathcal{T}} \in \mathcal{E} \rightarrow \mathcal{T}$ assigns processing times to events.*

Table I lists the notations used throughout the paper.

TABLE I
Summary of Symbol Notations

| Notation | Description |
|---|---|
| $I, R, A$ | Set of instances, resources, and activities |
| $WI$ | Set of work items |
| $wi_{i,k}$ | $k_{th}$ work item of instance $I_i$ |
| $p_{i,k,j}$ | Processing time of work item $w_{i,k}$ by $R_j$ |
| $ri_i, rr_j$ | Remaining time for $I_i$, $R_j$ to be ready |
| $S_i$ | Start time of $I_i$ |
| $F_i$ | Finish time of $I_i$ |
| $C_i$ | Completion time of $I_i$ ($C_i = F_i - S_i$) |
| $w_i$ | Weight of $I_i$ |

### B. Problem Statement

In this subsection, we define non-clairvoyant online job shop scheduling problem which we endeavor to solve using the two-phase method.

**Definition 4** (Non-clairvoyant Online Job Shop Scheduling)**.** *Given a set of instances I, where each instance has a set of operations which needs to be processed in a specific order, non-clairvoyant online job shop scheduling problem finds an optimal scheduling of all operations within instances while minimizing $\sum_i w_i C_i$, where $w_i$ is the weight of $I_i$ and $C_i$ is the difference between the finish time, $F_i$, and start time, $S_i$, of $I_i$. We also make some assumptions as follows.*

- *We are unaware of the data concerning an instance except for the weight of it.*
- *We only find out what the next operation of an instance is when an instance finishes its current operation.*
- *Each operation has a specific set of resources that it needs to be processed on and only one operation within an instance can be processed at a given time.*
- *An operation cannot be preempted, so once processing begins on an operation, it cannot be stopped until complete.*

## III. Baseline Approach and Insights

In the following, we describe a running example and a baseline approach called WeightGreedy. Also, we explain the insights from which we develop our suggested method.

### A. Running Example

Throughout this paper, a simple situation described in Fig. 1-(c) will serve as a running example. As shown in Fig. 1-(a), a node on the left means a work item, while a node on the right indicates a resource. An arc between nodes represents that a work item can be processed by a resource and its label means the processing time taken for the resource to finish the work item. Weights of instances are listed in Fig. 1-(b). Assume we are now at $T = t$. There are four work items (i.e., $wi_{1,1}(1_{st}$ work item of instance $I_1), wi_{2,2}, wi_{3,1}, wi_{4,1})$ and three resources (i.e., $r_1$, $r_2$, $r_3$) available at the moment. When allocating resource in this moment, we are unaware of a red value on an arc, (i.e., the processing time of a work item by a resource), a green value under a node (i.e., the start time

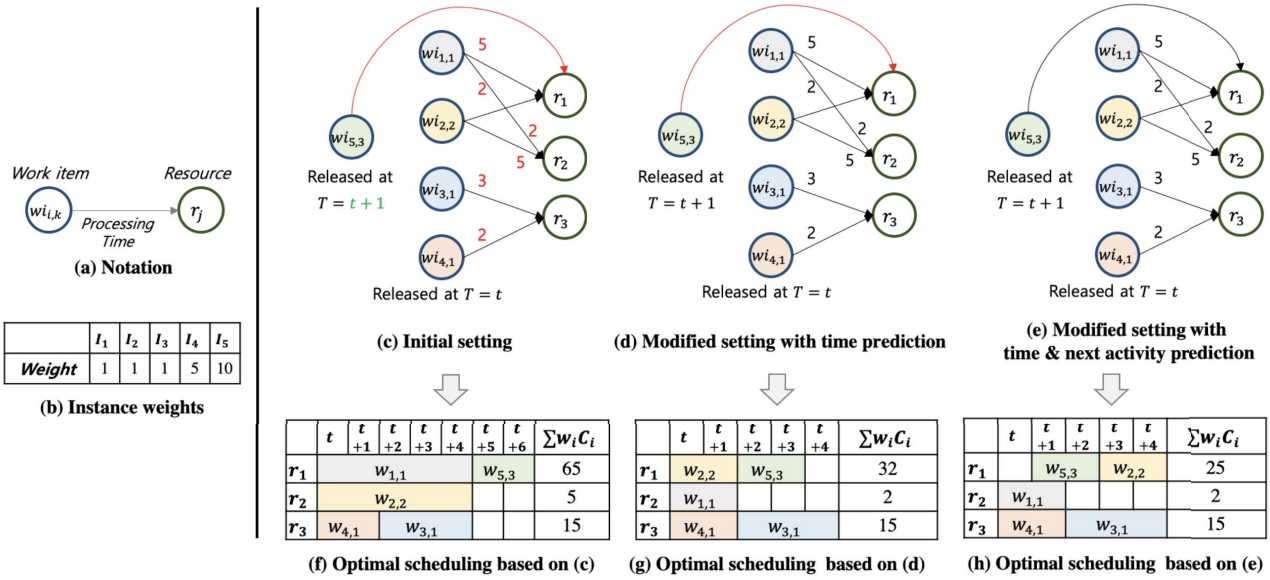



Fig. 1. An example of different problem settings and optimal schedules

of a new work item), and a red line between the left and right nodes (i.e., the resource requirement of a work item).

### B. A Baseline Approach

It is non-trivial to optimally assign resources in the above example since we only have limited information about the process. In this subsection, we first introduce a baseline solution called WeightGreedy. The main idea of WeightGreedy is that each work item is assigned to an available resource in a "first come, first served" manner. If there exist conflicting demands for the same resource, the work item with higher instance weight is served first. If the competing work items have the same instance weights, the tie is broken at random.

Fig. 1-(f) shows the optimal resource scheduling based on the baseline approach. Note that the scheduling is conducted under pull mechanism, i.e., work items are offered to available resources which can freely pick any of them [18]. There exist four work items, which are released at $T = t$ as described in Fig. 1-(c). Since both $wi_{1,1}$ and $wi_{2,2}$ requires $r_1$ and they have the same instance weight (i.e., $w_1 = w_2 = 1$), we randomly assign $wi_{1,1}$ to $r_1$. After that, $wi_{2,2}$ is assigned to $r_2$. Next, $r_3$, which has two demands, is allocated to $wi_{4,1}$ since $w_4$ (i.e., 5) is higher than $w_3$ (i.e., 1). At $T = t+1$, $wi_{5,3}$ has just released, and $wi_{3,1}$ remains in the waiting list since it was not assigned at $T = t$. As there are no resources available at the moment, those work items are staying in the waiting list. Consequently, $wi_{3,1}$ is processed at $T = t + 2$ and $wi_{5,3}$ at $T = t + 5$, as shown in Fig. 1-(f). The subtotal weighted completion times for resources are 65, 5, 15, resulting in a total sum of 85.

### C. Insights

First insight we find out is that we can improve the policy regarding the conflicting demands of a resource by predicting the processing times of running work items. As shown in Fig. 1-(d), it is better to assign $wi_{1,1}$ to $r_2$ and $wi_{2,2}$ to $r_1$ since these matches have lower processing time than the ones from the baseline approach, which then results in lower total weighted completion time. The optimal scheduling is described in Fig. 1-(g). The total weighted completion time is reduced to 59 from 85 due to the time predictions.

Secondly, we identify that we can further improve resource scheduling if we reserve resources for instances that have higher weights. As shown in Fig. 1-(e), if we know, in advance, that $r_1$ is required by $wi_{5,3}$ at $T = t+1$, we can reserve $r_1$ to serve it since $I_5$ has much higher weight than the others. Thus, our optimal solution, at $T = t$, is to make pseudo-assignment of $wi_{5,3}$ to $r_1$ and execute it right after $wi_{5,3}$ is released, i.e., at $T = t+1$. To figure out that $I_5$ requires $r_1$, we need first to predict the next activity of $I_5$ and then to find resources that can serve the activity, i.e., $r_1$ in this case. Resulting schedule is shown in Fig. 1-(h), where the total weight completion time of $r_1$ decreases from 32 to 25.

## IV. Method

This section proposes a two-phase method, which optimizes resource scheduling based on the time and next event prediction in the non-clairvoyant online setting. A general overview is presented first and, afterward, we explain each step in more detail.

### A. Overview

Our method consists of two phases: *offline prediction model construction* and *online resource scheduling*. Fig. 2 describes the overview of this method.

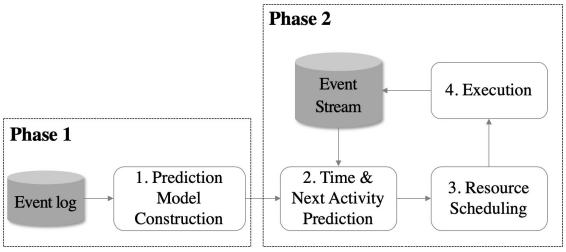



Fig. 2. Overview of two-phase method

*1) Phase 1: Offline Prediction Model Construction:* The first phase of our suggested method is to build prediction models both on the processing time and the next activity of ongoing instances based on an event log. It has been actively studied to predict the remaining time and the next activity of ongoing instances [6], [7], [19]. In this paper, we use one of the state-of-the-art methods suggested in [19], which deploys LSTM neural networks [20]. The model is trained on one-hot encoded vectors of historical traces which are obtained by transforming data from the event log.

*2) Online Resource Scheduling:* The second phase consists of three steps: *time and next event prediction*, *resource scheduling*, and *execution*. In this phase, optimal resource allocation takes place, based on the predictions on running instances whose information is recorded in event stream. First, we predict the processing time and the next activity of each running instance. For the predictions, we use the prediction model built in phase 1. Next, they are passed into the resource scheduling step, where a pseudo-assignment between a work item and a resource is established. Afterward, the pseudo-assignment is executed for any valid pairs of a work item and a resource. The event stream is updated every time the execution is carried out and this becomes an input for the next iteration of these three steps.

### B. Prediction Model Construction

In this step, we aim at building a time prediction function $f_t$ and a next activity prediction function $f_a$ such that $f_t(hd^k(\sigma)) = hd^1(tl^{k-1}(\pi_{\mathcal{T}}(\sigma)))$ and $f_a(hd^k(\sigma)) = hd^1(tl^k(\pi_{\mathcal{A}}(\sigma)))$, given $k$. Fig. 3 describes an architecture of our prediction model, which has two shared LSTM layers followed by two specialized layers for each task, i.e., time prediction and next activity prediction. An input is produced from an event log by transforming historical event $e \in hd^k(\sigma)$ with one-hot-encoding. In other words, each event $e_i \in hd^k(\sigma)$ is encoded as a vector of length $|A| + |R|$, such that $|A| + |R|$ features are all set to 0 except the one occurring at the index of $\pi_{\mathcal{A}}(e_i)$ and $\pi_{\mathcal{R}}(e_i)$.

Two target outputs of the model, i.e., processing time and next activity, are generated from the event log as well. The

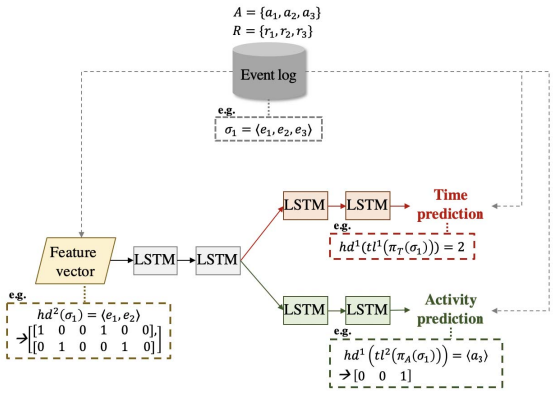



Fig. 3. An architecture of prediction model and a training example

first output, processing time, is a numerical value defined as $hd^1(tl^{k-1}(\pi_{\mathcal{T}}(\sigma)))$. The second output, next activity, is a categorical value represented as $hd^1(tl^k(\pi_{\mathcal{A}}(\sigma)))$. It is then encoded with one-hot-encoding as we do when generating a feature vector.

Suppose we train our model with $\sigma_1 = \{e_1, e_2, e_3\}$ and $k = 2$, shown in Fig. 3. The first two events of $\sigma_1$, i.e., $hd^2(\sigma_1) = \{e_1, e_2\}$, become a training input of out model after transforming them into a one-hot-encoded vector. Since we have $A = \{a_1, a_2, a_3\}$ and $R = \{r_1, r_2, r_3\}$, the length of the one-hot-encoded vector for each event, $e_1$ and $e_2$, is set to 6. As $\pi_{\mathcal{A}}(e_1) = a_1$, and a resource, $\pi_{\mathcal{R}}(e_1) = r_2$, the one-hot-encoded vector becomes $[1, 0, 0, 1, 0, 0]$. We can calculate the one-hot-encoded vector for $e_2$ in the same manner, i.e., $[0, 1, 0, 0, 1, 0]$. Our first target output is $hd^1(tl^1(\pi_{\mathcal{T}}(\sigma_1)))$, i.e., the processing time of $e_2$, 2. The second one is $hd^1(tl^2(\pi_{\mathcal{A}}(\sigma_1)))$, i.e., the activity of $e_3$, $a_3$, which is then transformed to $[0, 0, 1]$.

We train all sets of network weights using *Adam algorithm* [21] such that the mean absolute error (MAE) between the actual processing time and the predicted processing time, and the cross-entropy between the actual next activity and predicted next activity are minimized. All weights are initialized with *Xavier Initialization* [22]. We use 100 dimensions for the size of the LSTM memory. As regularization strategies, we use *Dropout* [23] and *Batch Normalization* [24].

### C. Time & Next Activity Prediction

Based on the prediction model we construct in the previous step, we predict the processing time and the next activity of ongoing instances from the event stream. Event stream provides information of running instances. Whenever a work item of an instance is initialized, we conduct two consecutive predictions regarding the instance. The first prediction aims at predicting the next operation of the instance. Based on this next activity prediction, we create artificial events ($\widehat{e}$) by combining the predicted next activity with available resources. Each of these artificial events, coupled with historical events, becomes an input for the second prediction. The purpose of

this prediction is to predict the processing time of the artificial event. Note that we iterate this second prediction for each artificial event. These prediction results are used to enhance the network as shown in Fig. 1-(e). The results are updated when the running instance is ready for the next operation only if the previous prediction result on next activity differs from the actual one.

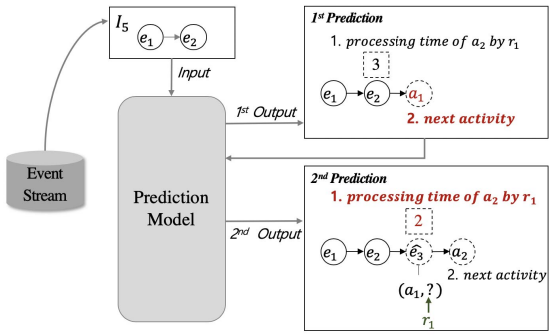



Fig. 4. An example of time and next activity prediction

For $I_5$ in Fig. 4, we conduct the first prediction based on the records from event stream. From the one-hot-encoded vector of $\{e_1, e_2\}$, the processing time of $e_2$ and the next activity $a_1$ are predicted. The next activity prediction result, $a_1$, is used to constitute an artificial event $\hat{e_3}$ with $r_1$ who is the only available resource for $a_1$. The predicted processing time of $a_1$ by $r_1$ is 2. These prediction results enable us to consider $I_5$ for optimal resource allocation at $T = t$ as described in Fig. 1-(e).

### *D. Resource Scheduling*

In this step, we solve a min-cost max-flow network problem, which aims at finding a maximum flow with the smallest possible cost, in order to optimize resource allocation. We start by constructing a bipartite graph such that nodes on the left are work items and those on the right are resources. Using the results from the previous step, we can include not only the work items and the resources in the queue but also those in process. For example, in Fig. 5, the graph includes a dotted circle, which represents a work item which is expected to appear later. The set of nodes on the left (right) is denoted by $\widehat{WI}$ ($\widehat{R}$). Afterward, we add an edge between a pair of nodes in the bipartite graph, i.e., $(wi, r)$ ($wi \in \widehat{WI}$, $r \in \widehat{R}$). Each edge contains $(cost, capacity)$, where $cost = (p_{i,k,j} + max(ri_i, rr_j, 0))/w_i$ such that $p_{i,k,j}$ is the processing time of work item $w_{i,k}$ by $r_j$, $ri_i$ is the remaining time for $I_i$ to be ready, and $rr_j$ is the remaining time for $r_j$. Note that we devise the cost function to optimize our objective (i.e., minimizing the total weighted completion time) by assigning less cost to edges which have less processing time and higher instance weight, while giving a penalty if a work item or a resource is not prepared. Finally, we adopt the minimum cost maximum flow algorithm based on network simplex method [25] to generate an optimal matching between work items and resources, i.e., pseudo-assignment.

Algorithm 1 illustrates the generation of a pseudo-assignment between work items and resources. Both predicted and prepared work items are instantiated to the set of nodes on the left, $\widehat{WI}$, while predicted and prepared resources are instantiated to the set of nodes on the right, $\widehat{R}$, in a bipartite graph. In lines 1-7, we create a source node and a sink node and then we add edges connecting source node to left nodes and right nodes to sink node. Each edge has a cost of zero and a capacity of one. In lines 8-12, we add edges between left nodes and right nodes if the work items (left) can be processed by resources (right), where the cost is calculated as $(p_{i,k,j} + max(ri_i, rr_j, 0))/w_i$ and the capacity is one.

**Algorithm 1** Resource scheduling algorithm

**Input:** $\widehat{WI}, \widehat{R}$
**Output:** Pseudo-Assignment $\widehat{M}$
1: *Produce source node s, sink node t*;
2: **for** *node* $wi_{i,k} \in \widehat{WI}$ **do**
3: *add edge* $(s, wi_{i,k}, (0, 1))$
4: **end for**
5: **for** *node* $r_j \in \widehat{R}$ **do**
6: *add edge* $(r_j, t, (0, 1))$
7: **end for**
8: **for** *node* $wi_{i,k} \in \widehat{WI}$ **do**
9: **for** *node* $r_j \in \widehat{R}$ **do**
10: $c \leftarrow (p_{i,k,j} + max(ri_i, rr_j, 0))/w_i$
11: *add edge* $(wi_{i,k}, r_j, (c, 1))$
12: **end for**
13: **end for**
14: $\widehat{M} \leftarrow MinCostMaxFlow(s, t)$
15: **return** $\widehat{M}$

In Fig. 5, $\widehat{WI}$ has five elements, one of which is in progress and four of which are ready for the assignment and all three elements of $\widehat{R}$ are ready for assignments. After running a min-cost max-flow algorithm, we have a pseudo-assignment of three matches represented as bold edges, i.e., $w_{5,3}$ *to* $r_1$, $w_{1,1}$ *to* $r_2$, $w_{4,1}$ *to* $r_3$.

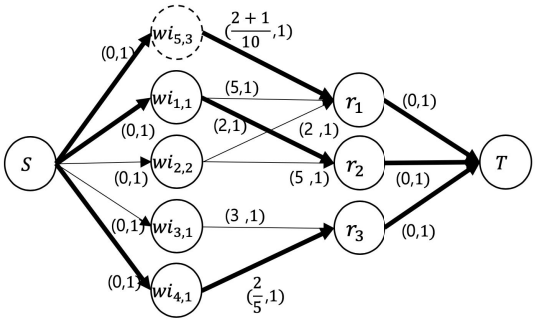



Fig. 5. An example of resource scheduling

### E. Execution

In this step, we explain how the pseudo-assignment is executed. The main idea is that we filter executable matches from the pseudo-assignment and execute them. An executable match means both corresponding instance and resource are available at the current time, while a non-executable match indicates that either of them is not available.

In Fig. 6-(a), the solid circle represents availability of an instance or resource, while the dotted circle means unavailability. Thus, there exist two executable matches, i.e., blue arcs, and one non-executable match, i.e., red arc. We filter only executable matches as shown in Fig. 6-(b) and execute those matches. In other words, $wi_{1,1}$ and $wi_{4,1}$ are processed by $r_2$ and $r_3$, respectively.

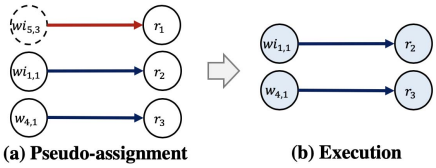



Fig. 6. An example of execution

## V. Evaluation

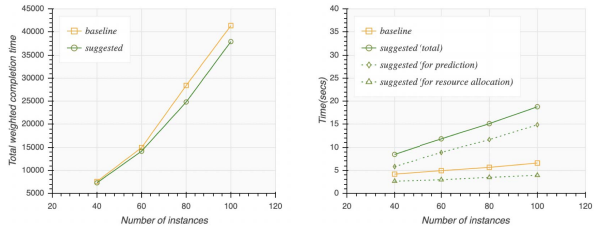
A twofold approach is used to evaluate the proposed method. First, we evaluate the ability of our proposed method to optimize resource allocation in the non-clairvoyant online setting using an artificial event log. Second, we discuss the application of the method on a real-life log.

### A. Artificial event log

*1) Experimental design:* Our proposed method is evaluated by comparing its ability to solve a non-clairvoyant online job shop problem with the baseline approach in terms of total weighted completion time. To this end, an artificial log is generated by simulating a simplified process at an emergency department of a hospital, composed of 11 activities and 25 resources. The department operates 24 hours a day. Each resource has his/her own set of activities that they are able to serve and the processing times of work items vary depending on the proficiency level. Patients with different weights come into the process in a regular interval. We assume a non-clairvoyant online environment, so we do not know how long the current operation of a patient will take before it finishes. In addition, we find out what the next operation of a patient will be only after it finishes its current operation.

We simulate this process for 7 days to produce an event log which will be used to construct a prediction model. For each activity, we define processing times of its available resources, which are assumed to follow *Gaussian* distribution. Each patient is assigned a weight ranging from 1 to 10 which is assumed to follow a uniform distribution. The resulting log has 1000 patients and 25 resources. We build an event stream by generating 100 patients entering the department in a regular interval for 6 hours. Each patient has his/her future activities which are unknown when making a schedule.

*2) Results:* Based on the event stream we generated, we compare our proposed method with the baseline, Weight-Greedy, in terms of total weighted completion time. Fig. 7 shows experimental results on varying number of instances(i.e., patients). The total weighted completion time increases with the increase of $|I|$ for both WeightGreedy and our suggested method Fig. 7-(a). The proposed method outperforms the baseline approach in all sizes of instances since it takes potential work items into account. For example, when the number of instances is 80, the total weighted completion time of the baseline approach (i.e., 28,393) is 14 percent higher than the one of the suggested method (i.e., 24,804). Both algorithms need more time when $|I|$ increases as shown in Fig. 7-(b). The computation time for the suggested method is relatively higher than the baseline approach, due to its prediction step which occupies most of the computations.




(a) Total weighted completion time of varying |I| (b) Computation time of varying |I|

Fig. 7. Experimental results on varying $|I|$

### B. Real-life event log

*1) Experimental design:* For the evaluation of our proposed method on real-life problem, we use an event log from Business Processing Intelligence Challenge 2012 (BPIC'12). It contains the data regarding the application procedure for a personal loan or overdraft at a global financing organization over a roughly six month period from October 2011 to March 2012. The dataset is comprised of a total of 262,200 events within 13,087 cases, recording the procedure from submitting an application to receiving a conclusion such as approval, cancellation, and rejection. This process is classified into three major types: one that refers to the states of the application itself, one that refers to the states of an offer, and one that tracks the states of work items that occur during the approval process. Among the three types, we investigate the third one containing events which are executed manually since we are not interested in the events performed automatically. Each case contains a case attribute, AMOUNT_REQ, which represents the amount requested in the application. Based on it, we create 10 equally spaced buckets, such that the bucket having a label of 1 contains the cases with the lowest amount and the bucket having a label of 10 includes the cases with the highest amount. Each case is given a weight according to the label of a bucket where it belongs.

We build an event log from the dataset by filtering it to include the events before 10/3/2012 and use it as an input of phase 1 in our method. The instances and resources, who appear on 10/3/2012 in the dataset, are extracted from the dataset, and they become components of the event stream which we use in phase 2. For those instances, we only consider operations that took place on 10/3/2012, to produce work items. The information, such as the activities and resources an instance has experienced before 10/3/2012, is also used to make predictions. The resulting number of instances is 110, and each instance has conducted 3 activities for the date on average.

*2) Results:* From the prediction model and the event stream, we simulate the resource allocation on 10/3/2012. Table II shows the results from both baseline approach and our proposed method. The total weighted completion time of baseline approach (i.e., 1,479) is 42 percent higher than the one of our suggested method (i.e., 1038). The massive difference comes from assigning the most efficient resource to the work item and reserving resources to enable them process instances having higher weights. The computation time is much higher in the proposed method. This is because there are many arcs between instances and resources, i.e., each work item has many resource options, which requires high computation when predicting the parameters (i.e., 110.1 secs out of 115.6 secs).

TABLE II
EXPERIMENTAL RESULTS ON REAL-LIFE EVENT LOG

| Method | Total weighted completion time | Computation time(secs) |
|---|---|---|
| Baseline | 1479 | 7.6 |
| Suggested | 1038 | 115.6 |

## VI. RELATED WORK

Our work is related to the research on predictive business process monitoring and job shop scheduling.

### A. Predictive Business Process Monitoring

The prediction tasks can mainly be classified into four different categories: time, risk probability, performance indicators, next event [5]. Among them, time and next event prediction provide valuable inputs for resource allocation [19].

*1) Time Prediction:* Most of the studies to predict time-related values focus on predicting the remaining time of running cases [5]. The first framework for this problem is suggested in [6] and many approaches have been developed based on it. Folino et al. [26] extends the technique described in [6] by clustering the log traces according to the corresponding context features. Polato et al. [7] further enhance the approach by adding machine learning model on additional attributes of events. These approaches, however, assume that the underlying process is stationary, which is not always true [7]. To mitigate this limitation, Tax et al. [19] developed a method to predict both time and next event with Long Short Term Memory network, where there is no need for an explicit representation of the process.

*2) Next Event Prediction:* Relatively few works have been done in next event prediction. Most works use an explicit process model representation such as Hidden Markov Model (HMM). Le et al. [27] proposes hybrid Markov models for predicting the next step in a process instance. If an instance reaches an unknown state, the model results in the prediction based on the most similar state by applying edit distance. Lakshmanan et al. [28] suggests a method that uses the instance-specific Probabilistic Process Models (PPM) where state transition probabilities for a Markov chain is derived from decision trees mined from case attributes. Recently, Evermann et al. [29] proposes a method based on LSTM neural networks with embedding as an encoding technique. Tax et al. [19] suggests an LSTM-based model with one-hot-encoding and multi-task learning to improve the prediction accuracy.

### B. Job Shop Scheduling

A job shop scheduling problem consists of a set of machines that perform operations on jobs, where each job has a processing order through the machines. There are several constraints on jobs and machines. While the machine sequence of the jobs is fixed, the problem is to find the job sequences on the machines which optimizes an objective (e.g. minimize the makespan) [12]. It is well known that the problem is NP-hard [13] and belongs to the most intractable problems [14].

A huge amount of literature on the scheduling of job shops has been published in the last decades [12]. The approaches for solving this problem can be categorized into three groups: dispatching, shifting bottleneck heuristic, and local search [12]. However, the application of these approaches to practical usages is somewhat limited because most of these techniques are not amenable to actual utilization in real job shops [30]. Dispatching rules, on the other hand, have been widely adopted in industry as they are computationally efficient, and robust to uncertainty [15].

More than 100 dispatching rules are suggested in the literature [31]. However, there's no rule which is applicable in non-clairvoyant online job shop problem where we do not know basic information for dispatching [16]. To the best of our knowledge, there is no attempt to utilize the prediction results to improve the applicability and efficiency of dispatching rules.

## VII. CONCLUSION

In this paper, we suggest a concrete method to improve a business process using results from predictive business process monitoring. We start by proposing a problem of online resource allocation in non-clairvoyant online environment. To address the problem, the key insight is to predict future behaviors of instances and resources in the process. Based on the insight, we devise a novel two-phase method, which integrates offline prediction model construction with online resource scheduling. For the prediction model, we adopt one of the state-of-the-art techniques based on LSTM neural networks to predict both the processing time and the next activity of a running instance. For online resource scheduling, we utilize the predictions generated from event stream to build a bipartite

graph, after which we solve a minimum cost and maximum flow problem. We verify the effectiveness and efficiency of the proposed method on both an artificial log and a log from BPIC'12. We anticipate our work as a leap on the road to more intelligent prediction-based process improvement techniques for a wide range of domains.

The proposed method has several limitations. First, our proposed method relies heavily on the performance of the prediction model. If a prediction for the processing time of a work item differs, we fail to match the work item to the most efficient resource. Second, the computation time is relatively higher than the baseline approach. The high computation comes from the execution of predictions for every newly-assigned instances, after which the search space for solving a network problem increases.

For future work, we will extend this two-phase method to achieve another goal such as minimizing the potential risks in the business process by predicting other relevant parameters and defining a relevant cost function of network arcs. Another direction for future work is to extend the proposed method by adopting advanced dispatching techniques.

## ACKNOWLEDGMENT

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2018-0-0144) supervised by the IITP (Institute for Information & communications Technology Promotion)

## REFERENCES

[1] W. M. P. van der Aalst, *Process Mining: Data Science in Action*, 2nd ed. Heidelberg: Springer, 2016.

[2] W. M. P. van der Aalst, M. L. Rosa, and F. M. Santoro, "Business process management - don't forget to improve the process!" *Business & Information Systems Engineering*, vol. 58, pp. 1–6, 2016.

[3] A. Metzger, P. Leitner, D. Ivanovi, E. Schmieders, R. Franklin, M. Carro, S. Dustdar, and K. Pohl, "Comparing and combining predictive business process monitoring techniques," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 45, no. 2, pp. 276–290, Feb 2015.

[4] M. de Leoni, W. M. P. van der Aalst, and M. Dees, "A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs," *Information Systems*, vol. 56, pp. 235–257, 2016.

[5] A. E. Mrquez-Chamorro, M. Resinas, and A. Ruiz-Cortes, "Predictive monitoring of business processes: A survey," *IEEE Transactions on Services Computing*, vol. 11, no. 6, pp. 962–977, Nov 2018.

[6] W. M. P. van der Aalst, M. Schonenberg, and M. Song, "Time prediction based on process mining," *Information Systems*, vol. 36, no. 2, pp. 450–475, 2011.

[7] M. Polato, A. Sperduti, A. Burattin, and M. D. Leoni, "Time and activity sequence prediction of business process instances," *Computing*, vol. 100, no. 9, pp. 1005–1031, Sep 2018.

[8] Z. Huang, W. M. P. van der Aalst, X. Lu, and H. Duan, "Reinforcement learning based resource allocation in business process management," *Data & Knowledge Engineering*, vol. 70, no. 1, pp. 127–145, 2011.

[9] W. Zhao and X. Zhao, "Process mining from the organizational perspective," in *Foundations of Intelligent Systems*, Z. Wen and T. Li, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 701–708.

[10] R. Conforti, M. de Leoni, M. L. Rosa, W. M. P. van der Aalst, and A. H. ter Hofstede, "A recommendation system for predicting risks across multiple business process instances," *Decision Support Systems*, vol. 69, pp. 1–19, 2015.

[11] D. Applegate and W. J. Cook, "A computational study of the job-shop scheduling problem," *INFORMS Journal on Computing*, vol. 3, pp. 149–156, May 1991.

[12] J. Baewicz, W. Domschke, and E. Pesch, "The job shop scheduling problem: Conventional and new solution techniques," *European Journal of Operational Research*, vol. 93, no. 1, pp. 1–33, 1996.

[13] R. Graham, E. Lawler, J. Lenstra, and A. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," in *Discrete Optimization II*, ser. Annals of Discrete Mathematics, P. Hammer, E. Johnson, and B. Korte, Eds. Elsevier, 1979, vol. 5, pp. 287–326.

[14] E. L. Lawler, J. K. Lenstra, A. H. R. Kan, and D. B. Shmoys, "Chapter 9 sequencing and scheduling: Algorithms and complexity," in *Logistics of Production and Inventory*, ser. Handbooks in Operations Research and Management Science. Elsevier, 1993, vol. 4, pp. 445–522.

[15] C. Tsiushuang, "Dispatching rules for manufacturing job-shop operations," *IFAC Proceedings Volumes*, vol. 26, no. 2, Part 4, pp. 975–978, 1993, 12th Triennal Wold Congress of the International Federation of Automatic control. Volume 4 Applications II, Sydney, Australia, 18-23 Jul.

[16] N. Megow, "Coping with incomplete information in scheduling — stochastic and online models," in *Operations Research Proceedings 2007*, J. Kalcsics and S. Nickel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 17–22.

[17] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, 3rd ed. Springer Publishing Company, Incorporated, 2008.

[18] A. Kumar, W. M. P. van der Aalst, and E. M. W. Verbeek, "Dynamic work distribution in workflow management systems: How to balance quality and performance," *Journal of Management Information Systems*, vol. 18, no. 3, pp. 157–193, Jan. 2002.

[19] N. Tax, I. Verenich, M. La Rosa, and M. Dumas, "Predictive business process monitoring with lstm neural networks," in *Advanced Information Systems Engineering*, E. Dubois and K. Pohl, Eds. Cham: Springer International Publishing, 2017, pp. 477–492.

[20] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–1780, Dec 1997.

[21] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, Dec 2014.

[22] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *Journal of Machine Learning Research - Proceedings Track*, vol. 9, pp. 249–256, Jan 2010.

[23] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, Jun 2014.

[24] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, pp. 448–456.

[25] W. H. Cunningham, "A network simplex method," *Mathematical Programming*, vol. 11, no. 1, pp. 105–116, Dec 1976.

[26] F. Folino, M. Guarascio, and L. Pontieri, "Discovering context-aware models for predicting business process performances," in *On the Move to Meaningful Internet Systems: OTM 2012*, R. Meersman, H. Panetto, T. Dillon, S. Rinderle-Ma, P. Dadam, X. Zhou, S. Pearson, A. Ferscha, S. Bergamaschi, and I. F. Cruz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 287–304.

[27] M. Le, B. Gabrys, and D. Nauck, "A hybrid model for business process event prediction," in *Research and Development in Intelligent Systems XXIX*, M. Bramer and M. Petridis, Eds. London: Springer London, 2012, pp. 179–192.

[28] G. T. Lakshmanan, D. Shamsi, Y. N. Doganata, M. Unuvar, and R. Khalaf, "A markov prediction model for data-driven semi-structured business processes," *Knowledge and Information Systems*, vol. 42, no. 1, pp. 97–126, Jan 2015.

[29] J. Evermann, J.-R. Rehse, and P. Fettke, "A deep learning approach for predicting process behaviour at runtime," in *Business Process Management Workshops*, M. Dumas and M. Fantinato, Eds. Cham: Springer International Publishing, 2017, pp. 327–338.

[30] X. Qiu and H. Y. Lau, "An ais-based hybrid algorithm with pdrs for multi-objective dynamic online job shop scheduling problem," *Applied Soft Computing*, vol. 13, no. 3, pp. 1340–1351, 2013, hybrid evolutionary systems for manufacturing processes.

[31] Y. L. Chang, T. Sueyoshi, and R. S. Sullivan, "Ranking dispatching rules by data envelopment analysis in a job shop environment," *IIE Transactions*, vol. 28, no. 8, pp. 631–642, 1996.

# Using Convolutional Neural Networks for Predictive Process Analytics

Vincenzo Pasquadibisceglie, Annalisa Appice, Giovanna Castellano, Donato Malerba
Department of Computer Science
University of Bari Aldo Moro
Bari, Italy
Email: [vincenzo.pasquadibisceglie,annalisa.appice,giovanna.castellano,donato.malerba]@uniba.it

***Abstract*—Predictive process monitoring has recently become one of the main enablers of data-driven insights in process mining. As an application of predictive analytics, process prediction is mainly concerned with predicting the evolution of running traces based on models extracted from historical event logs. This paper presents a process mining approach, which uses convolutional neural networks to equip the execution scenario of a business process with a means to predict the next activity in a running trace. The basic idea is to convert the temporal data enclosed in the historical event log of a business process into spatial data so as to treat them as images. To this purpose, every trace of the event log is first transformed into the set of its prefix traces (i.e. sequences of events that represent the prefix of a trace). These prefix traces are mapped into 2D image-like data structures. Created spatial data are finally used to train a Convolutional Neural Network, in order to learn a deep learning model capable to predict the next activity (i.e. the activity associated to the event occurring after the last event in the considered prefix trace). This predictive deep model can be employed as a powerful service to support participants in performing business processes since it guarantees a higher utilization by acting proactively in anticipation. Preliminary tests with two benchmark logs are carried out to investigate the viability of the proposed approach.**

**Keywords:** Predictive process analytics; Next activity prediction; Convolutional neural networks; Spatial data modeling.

## I. Introduction

Modern information systems that support complex business processes maintain significant amounts (event logs) of process execution data (traces). Process mining techniques are used to analyze data extracted from event logs and provide surprising insights for managers, system developers, auditors and end users [1]. Predictive process monitoring has recently become one of the main enablers of data-driven insights in process mining. Being able to predict the future behavior of a business process is an important business capability [2]. In particular, the effectiveness of a managerial decision-making response to variation in the environment may strongly depend on the extent to which it can reduce the impact of uncertainty through prediction.

As an application of predictive analytics in Business Process Management (BPM), business process prediction is mainly concerned with predicting the evolution of running traces based on patterns extracted from a historical event log. Example use cases include techniques to predict the next activity (and the timestamp of the activity) [2], [3], [4], [5], [6], as well as the completion cycle time until a trace is resolved [6], [7], [8]. Specifically, the prediction of the next activity can be considered to guarantee the higher utilization by acting proactively in anticipation, while the accurate estimate of the remaining cycle time can be used to minimize the deadline violation.

Existing predictive analytics for these tasks (e.g. [7], [9]) are often built on top of process discovery algorithms, which use models of formal languages (e.g. Petri nets, transition systems) to describe how logged processes are being executed. However, due to the unstructured form of real business processes, it may be difficult to define pre- and post-conditions for activities. In fact process discovery algorithms often generate spaghetti-like process models [10], which can be hard to apply for the predictive scope. On the other hand, with the increasing popularity of machine learning in predictive analytics, growing interest has arisen in using machine learning to analyze event logs and gain accurate insights into the future of a business process. Several machine learning techniques (e.g. parametric regression [8], predictive clustering tree inducer [3], [4], Naive Bayes classifier [11]) have been already investigated in various tasks of business process predictions. They learn process predictive patterns based on the sequence of the activities that have occurred in the running trace, the execution times of those activities and any other available trace or business process data that is available at runtime.

However the recent success in predictive deep learning research and development is also attracting attention in process mining [5], [6]. Deep learning is a specific type of machine learning using Deep Neural Networks (DNNs) as predictive models. DNNs are artificial neural networks, in which a large number of layers of neurons are interconnected with different weights and activation functions, in order to learn the hidden relationship between input and output. Intuitively, input data are fed to the first layer that generates different combinations of the input [12]. These combinations are fed to the subsequent layers in order to derive different representations of the input features. The weights on links between layers are adjusted according to backward propagation, depending on the distance or loss function between true output and the output calculated by the neural network. After the learning process via multiple layers, a better understanding and representation of distinguishable features can be derived by enhancing the

978-1-7281-0919-0/19/$31.00 ©2019 IEEE
DOI 10.1109/ICPM.2019.00028

prediction accuracy.

A wide number of DNNs architectures and related learning algorithms have been proposed in the literature [13]. However, previous studies in predictive process mining have mainly considered the Long Short-Term Memory (LSTM) architectures [14] due to their natural ability in delivering consistently high accuracy in the natural sequence modeling of a trace. Evermann et al. [5] have seminally applied LSTMs to predict the next activity in a trace. Tax et al. [6] have continued this study and investigated the accuracy of LSTMs in various process predictive tasks (next activity, time of next activity and completion time). These studies clearly indicate the potential for deep learning in predictive process mining.

In this paper, we continue the research in deep learning for predictive process mining by focusing on the task of predicting the next activity in a running trace. Although LSTMs are a very promising solution to sequence and time series related problems, there are various deep learning architectures, widely investigated in machine learning, which may be explored in alternative. In this study, we explore the use of Convolutional Neural Networks (CNNs) that represent a robust class of DNNs widely applied in computer vision and speech recognition [15], [16]. Recent literature indicates that CNNs promise very high accuracy whenever applied to image data embedding a clear spatial structure [17], [18], [19]. The success of CNNs is largely attributed to the use of local filtering and max-pooling in the network architecture. These operations enable the network to capture deeply the spatial structure of image data. By applying filters on spatially-neighboring "pixels" of input images, the network can detect spatial patterns (e.g. edges, shading changes, shapes, objects) which contribute to extract significant features from the input, thus gaining in predictive accuracy [20]. In general, CNNs can be successfully applied to data embedding some localized concepts, i.e. when data points close to each other share some correlations, such as pixels in images, words in texts or audio signals in speech [19], [21], [22]. Along with this idea, in this paper we propose the application of CNNs to non-visual sequential data coming from traces of event logs. The contributions of this work can be summarized as follows:

- A data engineering scheme is defined, in order to identify a spatial structure in the sequential order of trace data and transform temporal data into spatial data. In particular, 2D image-like data structures are generated from prefix traces of each trace stored in the historical event log of a business process. The prefix traces of a trace correspond to the prefix sequences of the trace, which can be extracted by considering $l$ initial events of the trace selected starting from the beginning of the trace. For each trace with length $L$ (i.e. a trace that encloses $L$ events), $L-1$ prefix traces can be extracted from it by considering prefix trace length $l$ that varies between 1 and $L-1$. The image-aware representation derived fro the prefix traces of event log data paves the way for investigating the performance of CNNs also in process mining.

TABLE I

A FRAGMENT OF AN EXAMPLE EVENT LOG. THE ACTIVITY DOMAIN COMPRISES 6 DISTINCT ACTIVITIES, THAT IS, REGISTER REQUEST (R), EXAMINE THOROUGHLY (ET), CHECK TICKET (CT), DECIDE (D), REJECT REQUEST (RR), ACCEPT REQUEST (AR). EACH EVENT IS LINKED TO A SPECIFIC TRACE. IT CORRESPONDS TO AN ACTIVITY OF THE ACTIVITY DOMAIN AND ITS CORRESPONDING TIMESTAMP.

| TraceId | Activity | Timestamp |
|---|---|---|
| 1 | Register request (R) | 2010-12-30:11:02 |
| 1 | Examine thoroughly (ET) | 2010-12-31:10:06 |
| 1 | Check ticket (CT) | 2011-01-05:15:12 |
| 1 | Examine thoroughly (ET) | 2011-01-06:11:06 |
| 1 | Check ticket (CT) | 2011-01-06:15:12 |
| 1 | Decide (D) | 2011-01-07:11:18 |
| 1 | Reject request (RR) | 2011-01-07:14:24 |
| 2 | Register request (R) | 2010-12-30:11:32 |
| 2 | Check ticket (CT) | 2010-12-30:12:12 |
| 2 | Examine causally (EC) | 2010-12-30:14:16 |
| 3 | ... | ... |

- A CNN architecture is trained on the image-aware representation of the historical event log, in order to learn a deep learning model capable to predict the next activity in a new running trace.
- A preliminary investigation on the viability of this deep process predictive analytic is performed considering two benchmark event logs. This study shows that this CNN predictive analytic may yield more accurate predictions than the LSTM counterpart described in [6].

The paper is organized as follows. The image-like data engineering is described in Section II. Section III describes a CNN architecture to predict the next activity in a running trace, while Section IV describes the event log, the experimental setup and discusses the relevant results. Finally, Section V draws some conclusions and outlines some future work.

## II. IMAGE-LIKE DATA ENGINEERING

The basic assumption is that an event log contains event information on activities executed for specific traces of a certain business process type, as well as their duration.

An *event* $\epsilon_i$ is characterized by two mandatory characteristics, that is, the event corresponds to an activity $\mathsf{act}_i$ in the activity domain $\mathbf{A}$ and has a timestamp $\mathsf{ts}_i$ representing date and time of occurrence[1]. The *activity domain* $\mathbf{A}$ is the set of $h$ distinct activities, which may appear in an event according to the model of the business process under consideration.

An *event log* $\mathcal{L}$ is a set of events, where each event is linked to a particular trace and globally unique.

A *trace* $\mathcal{T}$ represents the execution of a business process instance. It is a finite sequence of $l$ distinct events such that time is non-decreasing in the trace—i.e., $\mathsf{ts}_i \leq \mathsf{ts}_j$, $1 \leq i < j \leq l$. An *event log* $\mathcal{L}$ is a bag of traces. An example describing a fragment of an event log is reported in Table I.

A *prefix trace* is a sub-sequence of a trace starting from the beginning of the trace. Formally, let $\mathcal{T}$ be a trace made up of $l = |\mathcal{T}|$ events, a prefix trace $\mathcal{PT}_k$ is the sequence of the first $k$ consecutive events of $\mathcal{T}$ with $1 \leq k < l$. The *target*

[1]In this study the resource executing the activity is neglected.

for the *next activity* task associated with $\mathcal{PT}_k$ is $\mathsf{act}_{k+1}$, i.e, the activity corresponding to the event $\epsilon_{k+1} \in \mathcal{T}$. According to this definition, $l-1$ prefix traces can be extracted from a trace of length $l$. For example, Table II reports the prefix traces extracted from trace 1 of Table I.

As described in [23], [24], a prefix trace can be represented in both the activity perspective (frequency of activity, control-flow of activities) and the performance perspective (time performance). In this study, we account for both these perspectives, in order to define the 2D structure of a prefix trace image. Let us consider $\mathcal{PT}_k = \epsilon_1, \epsilon_2, \ldots, \epsilon_k$, that is the $k$-th prefix trace of trace $\mathcal{T}$. Each event $\epsilon_i$ is the pair $(\mathsf{act}_i, \mathsf{ts}_i)$ with $1 \leq i \leq k$. $\mathcal{PT}_k$ is represented as a 2D image $\mathcal{I}_k$ with size $k \times h$, where $k$ is the length of the prefix trace and $h$ is the size of the activity domain $\mathbf{A}$ of the event log that contains $\mathcal{T}$. The imagery rows of $\mathcal{I}_k$ represent the consecutive indexes of the events in $\mathcal{PT}_k$ sorted by timestamp (i.e. row $x$ corresponds to the $x$-th event $\epsilon_x$ in $\mathcal{PT}_k$). The imagery columns of $\mathcal{I}_k$ represent the distinct activities of the activity domain (i.e. column $y$ corresponds to the $y$-th activity $\mathbf{act_y}$ in $\mathbf{A}$). Every imagery pixel located at the $(x, y)$ row-column position of $\mathcal{I}_k$ is a 2-dimensional vector including the activity channel $A_k$ and the performance channel $P_k$, respectively. The activity channel $A_k(x, y)$ measures how many times activity $\mathbf{act_y}$ has occurred in prefix trace $\mathcal{PT}_k$ from the starting timestamp $\mathsf{ts}_1$ up to timestamp $\mathsf{ts}_x$. The performance channel $P_k(x, y)$ measures how long time (in days) has passed from the starting timestamp $\mathsf{ts}_1$ up until the latest occurrence of activity $\mathbf{act}_y$ has appeared in prefix trace $\mathcal{PT}_k$ before timestamp $\mathsf{ts}_x$. We consider the latest occurrence of the activity as we intend to base the prediction mainly on the activities performed recently. In this way, we are able to model how recently each activity has been actually observed in the current execution. For example, Figures 1(a) and 1(b) show the numeric values of the imagery pixels derived to represent the prefix trace (R,2010-12-30:11:02), (ET, 2010-12-31:10:06), (CT, 2011-01-05:15:12), (ET, 2011-01-06:11:06), (CT, 2011-01-06:15:12), (D,2011-01-07:11:18) reported in Table II in the activity and performance channels, respectively. Figures 1(c) and 1(d) show the display in gray-scale of these imagery representations along both channels.

In short, by resorting to the data engineering process described above, an event log is transformed into a labeled imagery dataset. For every trace in the event log, for every prefix trace in this trace, the 2D image generated to represent the prefix trace is labeled with the upcoming activity of the prefix trace. A CNN architecture can be trained over this imagery dataset as a business process analytic tailored to predict the next activity of a running trace. However, the use of this deep learning analytic poses a technical issue. All images should have the same size in the training set, while the number of rows of a prefix trace image may vary according to the prefix trace length. To address this issue, images are all generated with a fixed number of rows $K$. This number is estimated as the length of the longest prefix trace in the event log. In this way, a prefix trace with length $k < K$ is in any case converted into an image $K \times h$. As there is no actual information to generate pixels on rows $1, 2, \ldots K-k$, the unavailable values of the activity and performance channels are assigned to zero over these artificial imagery rows.

## III. Convolutional Neural Networks

CNNs extend simple fully connected feed-forward Neural Networks by introducing three extra concepts: local filters (convolution), pooling, and weight sharing. Typically a CNN consists of one or more pairs of convolutional and max pooling layers [15]. A convolutional layer applies a set of filters that are replicated along the whole input to process small local parts of the input. A pooling layer generates a lower resolution version of the convolutional layer output. This adds translation invariance and tolerance to minor differences of positions of patterns in the input. Higher layers use more broad filters that work on lower resolution inputs to process more complex parts of the input. Top fully connected layers finally combine inputs from all positions to do the classification of the overall inputs. This hierarchical organization enables a CNN to model local structures in the input using supervised learning algorithms.

The fist convolutional layer receives the input image $I(x, y)$ (input map) and convolves it by applying a set of $M$ filters. Each filter is a squared matrix of size $s$ containing weights $w_{j0}$ that acts as a local kernel on the input space, i.e. it can be modeled as a node that receives input only from a limited portion of the whole input (the receptive field of the node) and is thus suited to exploit local correlation hidden in the data. Each node $j$ applies the convolution operator over the input image, by computing the inner product of the filter at every location in the image, and outputs the result as a feature map $h_j(x, y)$. A non-linear function $f()$ is then applied to each feature map providing activations $a_j^{(l)} = f(h_j^{(l)}(x, y))$. For the first convolutional layer the values of the feature maps $h_j^{(1)}(x, y)$ are obtained by convolving the input map $I(x, y) = h^{(0)}(x, y)$ with the respective kernel $w_{j0}^{(1)}$ followed by a non-linear activation function $f()$:

$$h_j^{(1)}(x, y) = f\left( \sum_{(u,v) \in U} w_{j0}^{(1)} h^{(0)}(x+u, y+v) + b_j^{(1)} \right)$$

with $U = \{(u, v) \in \mathbb{N}^2 | 0 \leq u \leq s, 0 \leq v \leq s\}$.

Theoretically, any non-linear function can be used as activation function, provided that it is continuous and differentiable, as required by the backpropagation learning algorithm. The most common choices are tanh, logistic, softmax and relu. In our case we used the Relu function. $f(x) = max(0, x)$.

A pooling or sub-sampling layer [25] is added on top of each convolutional layer to achieve spatial invariance and to reduce the dimensionality of the feature maps, preserving important information and discarding irrelevant details. The pooling operation is typically a sum, average, maximum or even a combination of various methods. In our study we used the max pooling (that takes the maximum value of filter activation from different positions within a specified window) since it presents the best results in the literature [26].

TABLE II
PREFIX TRACES EXTRACTED FROM TRACE 1 REPORTED IN TABLE I.

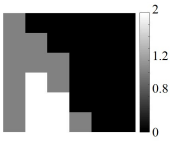
| TraceId | PrefixTraceId | Prefix trace | Next activity (target) |
|---|---|---|---|
| 1 | 1 | (R,2010-12-30:11:02) | ET |
| 1 | 2 | (R,2010-12-30:11:02),(ET,2010-12-31:10:06) | CT |
| 1 | 3 | (R,2010-12-30:11:02),(ET,2010-12-31:10:06),(CT,2011-01-05:15:12) | ET |
| 1 | 4 | (R,2010-12-30:11:02),(ET,2010-12-31:10:06),(CT,2011-01-05:15:12),(ET,2011-01-06:11:06) | CT |
| 1 | 5 | (R,2010-12-30:11:02),(ET,2010-12-31:10:06),(CT,2011-01-05:15:12),(ET,2011-01-06:11:06), (CT,2011-01-06:15:12) | D |
| 1 | 6 | (R,2010-12-30:11:02),(ET,2010-12-31:10:06),(CT,2011-01-05:15:12)(ET,2011-01-06:11:06), (CT,2011-01-06:15:12), (D,2011-01-07:11:18) | RR |

| Time/Activity | R | ET | CT | D | RR | AR |
|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 1 | 0 | 0 | 0 | 0 |
| 3 | 1 | 1 | 1 | 0 | 0 | 0 |
| 4 | 1 | 2 | 1 | 0 | 0 | 0 |
| 5 | 1 | 2 | 2 | 0 | 0 | 0 |
| 6 | 1 | 2 | 2 | 1 | 0 | 0 |

(a) Activity channel-matrix

| Time/Activity | R | ET | CT | D | RR | AR |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0,9611 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0,9611 | 6,1736 | 0 | 0 | 0 |
| 4 | 0 | 7,0002 | 6,1736 | 0 | 0 | 0 |
| 5 | 0 | 7,0020 | 7,1736 | 0 | 0 | 0 |
| 6 | 0 | 7,0002 | 6,1732 | 8,0111 | 0 | 0 |

(b) Performance channel-matrix




(c) Activity channel-display


(d) Performance channel-display

Fig. 1. The 2D image-like representation of prefix trace (R,2010-12-30:11:02), (ET, 2010-12-31:10:06), (CT, 2011-01-05:15:12), (ET, 2011-01-06:11:06), (CT, 2011-01-06:15:12), (D,2011-01-07:11:18): activity channel (data matrix in Figure 1(a) and gray-scale display in Figure 1(c)) and performance channel (data matrix in days in Figure 1(b) and gray-scale display in Figure 1(d)).

On the overall, the CNN consists of one or more pairs of convolutional and max-pooling layers, where the lowest layers process a small number of input maps so as to generate higher level representations with lower resolution. The size of maps decreases in higher layers. In general, the convolution map $j$ in layer $l$ can have several kernels $w_{(ji)}^{(l)}$ operating on different maps $i$ in the preceding layer $l-1$. Thus, the general activation formula for the convolution map $j$ in layer $l$ is:

$$h_j^{(l)}(x,y) = f\left(\sum_{i\in I}\sum_{(u,v)\in U} w_{ji}^{(l)} h_i^{(l-1)}(x+u, y+v) + b_j^{(1)}\right)$$

where $I$ is the set of maps of the preceding layer to which the convolution map $j$ is connected. In our model, three pairs of convolutional and max-pooling layers are considered, leading to the hierarchical architecture depicted in fig. 2.

The final layers in the CNN are fully connected just like that of a normal feed-forward NN. These fully connected layers combine different local structures extracted in the lower layers in order to perform the final classification/prediction task. For the $H$ nodes in the output layer the softmax is usually chosen as activation function:

$$o_j = f(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{H} e^{x_k}}$$

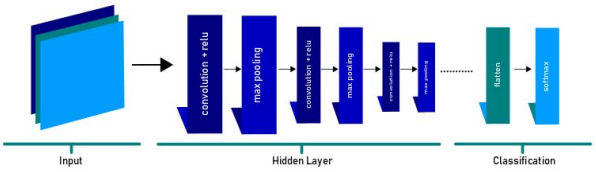



Fig. 2. CNN architecture

The training of CNNs is very similar to the training of other types of NNs, such as Multilayer Perceptrons (MLP). A set of labeled training examples $\left\{(\mathcal{I}_p, \mathbf{t}_p)\right\}_{p=1}^{P}$ is required, where $\mathcal{I}_p$ is the $p$-th training image and $\mathbf{t}_p$ is the desired target output (in our case $\mathbf{t}_p$ represents the correct next activity to be predicted). Training consists in an optimization process aimed at minimizing a loss functions representing the overall error of the network on the training data. The most used training algorithm is the online Backpropagation with a real time weight update [27]. However, many other optimization methods, mainly based on gradient descent optimization have been proposed in the literature [28]. These algorithms are

similar to algorithms used to train the MLPs [29], the only difference to take into account is the weight sharing in the convolutional and sub-sampling layers. Using the training algorithm the weights of the network are adjusted so as to minimize the loss function.

## IV. Experimental results

The presented process predictive analytic, called ImagePP-Miner (Image-like Predictive Process Miner), is implemented in Python 3.6 64 bit version. It is evaluated for the predictive task at hand (next activity). Specifically, the evaluation study aims to investigate the viability of the idea of transforming temporal event data into spatial data, verify the effectiveness of the proposed image-like representation of trace data in process mining and compare the accuracy of the CNN architecture to that of the LSTM competitor described in [6].

### A. Data

Two benchmark event logs also used in [6] are considered:

- Helpdesk event log[2] is a benchmark event log that contains events from a ticketing management process of the help desk of an Italian software company. The business process consists of 9 activities. As reported in [6], an artificial final event is added to each trace; this is associated with the artificial activity denoted as $\mathsf{end}$. All traces start with the insertion of a new ticket into the ticketing management system. Each trace ends when the issue is resolved and the ticket is closed. This log contains 3804 traces and 13710 events. The length of the traces varies between 1 and 14.
- BPI challenge 2012 (BPI12_W_Complete) event log[3] pertains to an application process for a personal loan or overdraft within a global financing organization. The application process consists of three subprocesses: one that tracks the state of the application, one that tracks the states of work items associated with the application, and a third one that tracks the state of the other. Based upon considerations discussed in [6], in the context of predicting the coming events and their timestamps, there is no interest towards events that are performed automatically. Thus, the evaluation is narrowed down to the 9658 work item traces, which contains events that are manually executed. In addition, only 72413 events of 6 activities with type complete are retained. This pre-processing is performed as in [6] to enable a comparison with competitor results. The length of the pre-processed traces varies between 1 and 74.

### B. Experimental settings

*a) Experimental methodology and evaluation metrics:* The experimental setting described in [6] is adopted in this study to make safe the comparison. For each event log, the first ordered 2/3 of the traces are used as training event log, while the accuracy of the activity predictions is evaluated on the remaining 1/3 of the traces. The next activity is evaluated on all prefix traces (running traces) of all testing traces. We note that similarly to [6], we do not resort to the traditional k-fold cross validation as a validation methodology for this study. In fact we would preserve a temporal ordering between training and testing traces. In this way, we simulate the real case where the predictive model is trained from historical data and its performance is evaluated on future data.

Figures 3(a) and 3(b) show the distribution of the (next activity) class in both the training and testing set for Helpdesk and BPI12_W_Complete, respectively. Figure 3(a) highlights that, as the activity $\mathsf{act}_3$ appears only at the beginning of few traces of Helpdesk, there is no prefix trace labeled with this activity in both the training and testing set of Helpdesk. We also note that both figures include class "end", that represents the class to predict when the trace has been completed. We considered two empirical scenarios:

- Scenario1: the class set includes the log activities plus mark "end" (10 classes in Helpdesk and 7 classes in BPI12_W_Complete)
- Scenario2: the class set that includes only log activities (9 classes in Helpdesk and 6 classes in BPI12_W_Complete).

In both scenarios, as reported in [6] no prediction is performed for 1-sized prefix traces as insufficient data are available for these prefixes to base the prediction upon. The time spent in seconds to train the CNN predictive model is computed, in order to analyze the efficiency of the learning process. The accuracy of the next activity prediction is computed, in order to measure the performance of the process prediction approach.

*b) CNN architecture and training:* The architecture of the CNN considered to process Helpdesk includes three pairs of convolutional and max-pooling layers. Each convolutional layer is configured as follows:

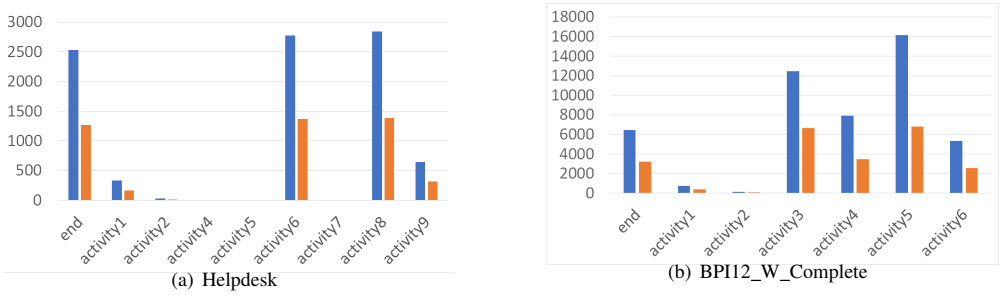
1) layer 1: #filters: 32; filter size: $2 \times 2$
2) layer 2: #filters: 64; filter size: $4 \times 4$
3) layer 3: #filters: 128; filter size: $8 \times 8$

Each pooling layer had a standard stride of 2 and a $2 \times 2$ sliding window. The feature maps produced by the three alternating layers of convolutional and max pooling filters are finally flattened and sent to a dense output layer of softmax units. The architecture structure is trained on log BPI12_W_Complete except for layer 3 that is omitted.

To speed-up the training of the CNN we apply the Batch Normalization [30] which is an additional step consisting in normalizing the output of each convolution layer before entering in the subsequent max-pooling layer. This makes the training process less sensitive to weight initialization and speeds up training, without increasing overfitting.

The CNN training is based on optimization of the categorical cross-entropy loss function. To minimize the loss functions we used the Adam (Adaptive moment estimation) optimizer since it has been suggested to be the best overall choice among state-of-art gradient descent optimization algorithms [28]. The

[2]https://data.mendeley.com/datasets/39bp3vv62t/1
[3]http://www.win.tue.nl/bpi/2012/challenge




(a) Helpdesk (b) BPI12_W_Complete

Fig. 3. Histogram of class (next activity) distribution in both training set (blue bar) and testing set (orange bar).

Adam method performs efficient stochastic optimization by computing only first-order gradients with little memory requirement [31]. Updates of network weights are computed iteratively (with batch size 128) on the basis of a running average of first and second moment of the gradient. Hyper-parameters of Adam optimizer are:

- the exponential decay rate $\beta_1 \in [0, 1)$ for the first moment estimates. We used default value $\beta_1 = 0.9$.
- the exponential decay rate $\beta_2$ for the second moment estimates. We use default value $\beta_2 = 0.999$.
- the learning rate (or stepsize) $\alpha$ that indicates the proportion that weights are updated. We fix $\alpha = 2 \times 10^{-4}$;
- a small tolerance $\epsilon$ to prevent any division by zero in the implementation. We fix $\epsilon = 10^{-8}$.
- a decay factor $\lambda$ to control the decay of the stepsize $\alpha$. We fix $\lambda = 4 \times 10^{-4}$.

To avoid overfitting, a validation set (20% of the training data) is considered in order to perform early stopping. The training stage stops when the loss on the validation set does not improve for a fixed number of epochs. For each scenario, the training phase is repeated on five trials so that the performance can be measured as the average (and standard deviation) on these trials.

### C. Results and discussion

We start by analyzing the convergence and efficiency of the learning process. Figure 5 depicts the trend of the loss function on the training and validation set during the training of the CNN with baseline early stopping size set equal to 6. For each log and scenario, we plot the loss of the best trial selected with respect to the accuracy performance. We note that the learning algorithm, in both logs and scenarios, converges in a few epochs to a small value of the loss function. Additional considerations are also deserved by the analysis of the computation time spent in seconds completing the training phase that was run on 8 CPU AMD Opteron 63xx, 16GB Ram Memory, Ubuntu 18.04.1 LTS. Results reported in Table III show that the training is commonly completed in a short training time (less than 5 minutes in Helpdesk and less than 26 minutes in BPI12_W_Complete).

We proceed this evaluation by analyzing the accuracy performance of the trained model on the testing set. To this

TABLE III
COMPUTATION TIME (IN SECONDS) ANALYSIS

| | Scenario1 | Scenario2 |
|---|---|---|
| Helpdesk | $204.6 \pm 51.36$ | $110.0 \pm 27.61$ |
| BPI12_W_Complete | $1461.6 \pm 356.12$ | $1530.0 \pm 199.22$ |

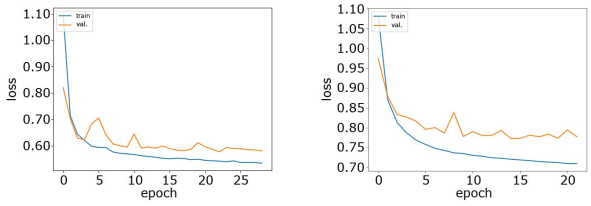



Fig. 4. Scenario1: Trend of the loss function during training of the CNN.

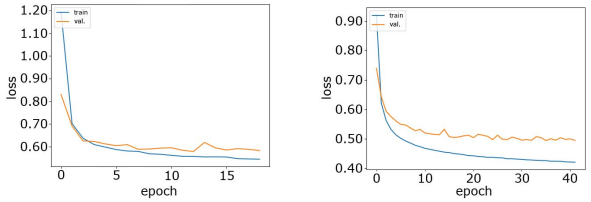



Fig. 5. Scenario2: Trend of the loss function during training of the CNN.

purpose, we consider the overall accuracy computed as the percentage of correctly predicted activities on the number of predictions. As we would investigate the sensitivity of the accuracy to the early stopping size, we collect results by varying early stopping size between 3, 6 and 9. Results collected in Table IV confirm that the highest accuracy is always achieved with the baseline early stopping size set equal to 6. We also note that, independently on the early stopping size, the proposed deep learning model achieves the highest accuracy in the Scenario1 for Helpdesk, while it achieves the highest accuracy in the Scenario2 for BPI12_W_Complete.

To deepen this accuracy analysis, we also consider the confusion matrix of the prediction accuracy and compute precision and recall to evaluate the accuracy performance achieved on each class/activity. Precision and recall metrics are collected in

TABLE IV
ACCURACY ANALYSIS: OVERALL ACCURACY

| | Scenario1 | | Scenario2 | |
|---|---|---|---|---|
| | Helpdesk | BPI12_W_Complete | Helpdesk | BPI12_W_Complete |
| early_stopping3 | 0.8062±0.0124 | 0.6835±0.0300 | 0.7243±0.0067 | 0.7810±0.0017 |
| early_stopping6 | 0.8182±0.0013 | 0.6859±0.0012 | 0.7393±0.0038 | 0.7817±0.0016 |
| early_stopping9 | 0.8169±0.0052 | 0.6800±0.0025 | 0.7312±0.0122 | 0.7788±0.0063 |

Table V for each class. These class-by-class analysis confirms that the completion of the trace is predicted with high precision and recall in Helpdesk, while the precision and recall are lower when we consider the ability of predicting the completion of the trace (class "end") in BPI12_W_Complete. This instability in the performance of predictive model suggests that the of the use of a CNN architecture in the prediction of the completion of a trace requires additional investigation. On the other hand, this detailed analysis of precision and recall, as it is performed at the class granularity, also highlights that although the classification model exhibits higher ability when used to predict activities, which are sufficiently represented in the training set (e.g. activity 6 and activity 8 of Helpdesk; activity 3, activity 4 and activity 5 of BPI12_W_Complete), its accuracy decreases drastically when it is evaluated with respect to minority classes (i.e.classes for which a few samples is available in the training set).

Finally, we complete this study by analyzing the overall performance of CNN learned by ImagePPMiner to that of the deep competitor, based on LSTM and described in [6]. Results are reported in Table VI for Scenario2, that is the same scenario considered [6]. They shows that ImagePPMiner achieves an overall classification accuracy of 73.93% in Helpdesk and 78.17 in BPI12_W_Complete, respectively. In both logs, CNN provides an accuracy gain in comparison to the LSTM model that achieves a classification accuracy of 71.23% in Helpdesk and 76.00% in BPI12_W_Complete. These results confirm that transforming temporal event data into spatial data may actually construct a reasonable representation of trace data, which may be used for accurate predictive analytics. In addition results in this study show that the use of CNN architecture is a promising predictive analytic enabler also in process mining, as it allows us to yield more accurate predictions of the next activity than the LSTM architecture.

## V. CONCLUSION

This paper describes a deep learning approach in predictive process mining, in order to yield accurate predictions of the next activity in a running trace. The foremost contributions of our work are:

- The investigation of a new process data engineering scheme, which transforms temporal event data into spatial data so as to treat them as images.
- The use of a CNN architecture to learn a deep learning model from the spatial representation of the prefix traces stored in the historical event log of business process.
- The evaluation of the accuracy of a deep learning analytic in the prediction of the next activity of a running trace.

The empirical study shows that the generation of 2D image-like data structures from historical prefix traces is a valid means to model trace data in both the activity and performance perspectives. In addition, the described process data engineering scheme allows us to use a CNN architecture as a process predictive analytic capable to yield more accurate predictions of the next activity than the LSTM architecture. Nevertheless, experimental results showed that the proposed system is not able to properly predict non-frequent activities. We are currently working to increase the accuracy of our system in minor activity prediction by introducing oversampling techniques in order to better represent rare activities in the log. In addition, we would explore the viability of a sliding window model to be used in combination with the proposed data engineering scheme. A sliding window will allow us to transform only window-selected temporal data in spatial data structure. The window size will also allow us to diminish the image size by limiting the risk of images filled with high number of blacks (zero values), while forcefully learning such data might lead to model overfitting and uninformative patterns learning. As additional future work, we plan to improve the empirical investigation by considering new event logs in various application domains (e.g. worker training) and various configurations of the CNN architecture. Moreover, we plan to study how our approach would scale with event logs with a much larger number of activities. Finally, we would investigate the extensions of the proposed approach to other process predictive tasks such as prediction of the timestamp of the next activity, completion cycle time and outcome of the execution.

## ACKNOWLEDGMENT

The research is partially supported by the POR Puglia FESR-FSE 2014-2020 - Asse prioritario 1 - Ricerca, sviluppo tecnologico, innovazione - Sub Azione 1.4.b BANDO INNOLABS - SOSTEGNO ALLA CREAZIONE DI SOLUZIONI INNOVATIVE FINALIZZATE A SPECIFICI PROBLEMI DI RILEVANZA SOCIALE - *Research project KOMETA (Knowledge Community for Efficient Training through Virtual Technologies)*, funded by Regione Puglia. The research of Vincenzo Pasquadibisceglie is funded by PON RI 2014-2020 - Big Data Analytics for Process Improvement in Organizational Development - CUP H94F18000260006. The computational work has been executed on the IT resources of the ReCaS-Bari data center, which have been made available by two projects financed by the MIUR (Italian Ministry for Education, University and Re-search) in the "PON Ricerca e Competitivita' 2007-2013" Program: ReCaS (Azione I - Interventi di

TABLE V
ACCURACY ANALYSIS: PRECISION AND RECALL

| Scenario1 | | | | | | Scenario2 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Helpdesk | | | BPI12_W_Complete | | | Helpdesk | | | BPI12_W_Complete | | |
| | Precision | Recall | | Precision | Recall | | Precision | Recall | | Precision | Recall |
| End | 0.9230±0.0016 | 0.9946±0.0018 | End | 0.3780±0.0042 | 0.2271±0.0415 | Act1 | 0.0 | 0.0 | Act1 | 0.5486±0.0084 | 0.2607±0.019 |
| Act1 | 0.0 | 0.0 | Act1 | 0.4654±0.0089 | 0.2162±0.0138 | Act2 | 0.0 | 0.0 | Act2 | 0.5832±0.0612 | 0.3029±0.0629 |
| Act2 | 0.0 | 0.0 | Act2 | 0.4272±0.0757 | 0.2042±0.0234 | Act4 | 0.0 | 0.0 | Act3 | 0.7931±0.0006 | 0.9935±0.0005 |
| Act4 | 0.0 | 0.0 | Act3 | 0.7239±0.0035 | 0.9557±0.0076 | Act5 | 0.0 | 0.0 | Act4 | 0.7555±0.0189 | 0.8692±0.0447 |
| Act5 | 0.0 | 0.0 | Act4 | 0.7216±0.0225 | 0.8191±0.0256 | Act6 | 0.7453±0.0065 | 0.9403±0.0227 | Act5 | 0.8294±0.0014 | 0.8117±0.002 |
| Act6 | 0.7466±0.0069 | 0.8988±0.0150 | Act5 | 0.7518±0.0024 | 0.8104±0.0027 | Act7 | 0.0 | 0.0 | Act6 | 0.5623±0.0263 | 0.2219±0.0629 |
| Act7 | 0.0 | 0.0 | Act6 | 0.4000±0.0107 | 0.1737±0.0215 | Act8 | 0.7278±0.0517 | 0.7115±0.05803 | - | - | - |
| Act8 | 0.7353±0.0390 | 0.6793±0.0443 | - | - | - | Act9 | 0.3666±0.1639 | 0.006±0.0027 | - | - | |
| Act9 | 0.175±0.1859 | 0.0081±0.0138 | - | - | - | - | - | - | - | - | - |

TABLE VI
ACCURACY ANALYSIS: CNN VS LSTM MODEL

| Classifier | Overall accuracy | |
|---|---|---|
| Helpdesk | LSTM-based model [6] | 71.23% |
| | CNN-based model (ImagePPMiner) | 73.93% |
| BPI12_W_Complete | LSTM-based model [6] | 76.00% |
| | CNN-based model (ImagePPMiner) | 78.17% |

rafforzamento strutturale, PONa3_00052, Avviso 254/Ric) and PRISMA (Asse II - Sostegno all'innovazione, PON04a2_A).

## REFERENCES


[1] W. M. P. van der Aalst, *Process Mining - Data Science in Action, Second Edition*. Springer, 2016. [Online]. Available: https://doi.org/10.1007/978-3-662-49851-4

[2] D. Breuker, M. Matzner, P. Delfmann, and J. Becker, "Comprehensible predictive models for business processes," *Journal MIS Quarterly*, vol. 40, pp. 1009–1034, 2016.

[3] S. Pravilovic, A. Appice, and D. Malerba, "Process mining to forecast the future of running cases," in *New Frontiers in Mining Complex Patterns - Second International Workshop, NFMCP 2013, Revised Selected Papers*, ser. LNCS, A. Appice, M. Ceci, C. Loglisci, G. Manco, E. Masciari, and Z. W. Ras, Eds., vol. 8399. Springer, 2014, pp. 67–81.

[4] A. Appice, D. Malerba, V. Morreale, and G. Vella, "Business event forecasting," in *10th International Forum on Knowledge Asset Dynamics, IFKAD 2015*, J. Spender, G. Schiuma, and V. Albino, Eds. Inst Knowledge Asset Management, 2015, pp. 1442–1453.

[5] J. Evermann, J.-R. Rehse, and P. Fettke, "A deep learning approach for predicting process behaviour at runtime," in *Business Process Management Workshops*, M. Dumas and M. Fantinato, Eds. Springer, 2017, pp. 327–338.

[6] N. Tax, I. Verenich, M. La Rosa, and M. Dumas, "Predictive business process monitoring with LSTM neural networks," in *International Conference on Advanced Information Systems Engineering*, E. Dubois and K. Pohl, Eds. Springer, 2017, pp. 477–492.

[7] W. M. P. van der Aalst, M. H. Schonenberg, and M. Song, "Time prediction based on process mining," *Information Systems*, vol. 36, no. 2, pp. 450–475, 2011.

[8] B. F. van Dongen, R. A. Crooy, and W. M. P. van der Aalst, "Cycle time prediction: When will this case finally be finished?" in *On the Move to Meaningful Internet Systems: OTM 2008*, R. Meersman and Z. Tari, Eds. Springer, 2008, pp. 319–336.

[9] A. Rogge-Solti and M. Weske, "Prediction of remaining service execution time using stochastic petri nets with arbitrary firing delays," in *ICSOC*. Springer, 2013, pp. 389–403.

[10] W. M. P. van der Aalst, "Process mining: discovering and improving spaghetti and lasagna processes," in *IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2011*, 2011, pp. 1–7.

[11] M. Ceci, M. Spagnoletta, P. F. Lanotte, and D. Malerba, "Distributed learning of process models for next activity prediction," in *Proceedings of the 22nd International Database Engineering & Applications Symposium, IDEAS 2018, V*, B. C. Desai, S. Flesca, E. Zumpano, E. Masciari, and L. Caroprese, Eds. ACM, 2018, pp. 278–282.

[12] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[13] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] Y. LeCun, Y. Bengio *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[16] B. Prasad and S. M. Prasanna, *Speech, audio, image and biomedical signal processing using neural networks*. Springer, 2007, vol. 83.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[18] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[19] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Transactions on audio, speech, and language processing*, vol. 22, no. 10, pp. 1533–1545, 2014.

[20] Y. LeCun, F. J. Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," in *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, vol. 2. IEEE, 2004, pp. II–104.

[21] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in neural information processing systems*, 2015, pp. 649–657.

[22] M. Alhussein, G. Muhammad, and M. S. Hossain, "Eeg pathology detection based on deep learning," *IEEE Access*, 2019.

[23] M. Song, C. Ganther, and W. Aalst, "Trace clustering in process mining," in *Business Process Management Workshops*, ser. Lecture Notes in Business Information Processing. Springer, 2009, vol. 17, pp. 109–120.

[24] A. Appice and D. Malerba, "A co-training strategy for multiple view clustering in process mining," *IEEE Trans. Services Computing*, vol. 9, no. 6, pp. 832–845, 2016.

[25] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *Artificial Neural Networks–ICANN 2010*. Springer, 2010, pp. 92–101.

[26] ——, "Evaluation of pooling operations in convolutional architectures for object recognition," in *Artificial Neural Networks–ICANN 2010*. Springer, 2010, pp. 92–101.

[27] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[28] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[29] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Neurocomputing: Foundations of research," *ch. Learning Representations by Back-propagating Errors*, pp. 696–699, 1988.

[30] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

# Predictive Performance Monitoring of Material Handling Systems Using the Performance Spectrum

Vadim Denisov
Eindhoven University of Technology
The Netherlands
Email: v.denisov@tue.nl

Dirk Fahland
Eindhoven University of Technology
The Netherlands
Email: d.fahland@tue.nl

Wil M.P. van der Aalst
Department of Computer Science,
RWTH Aachen, Germany,
Email: wvdaalst@pads.rwth-aachen.de

***Abstract*—Predictive performance analysis is crucial for supporting operational processes. Prediction is challenging when cases are not isolated but influence each other by competing for resources (spaces, machines, operators). The so-called performance spectrum maps a variety of performance-related measures within and across cases over time. We propose a novel prediction approach that uses the performance spectrum for feature selection and extraction to pose machine learning problems used for performance prediction in non-isolated cases. Although the approach is general, we focus on material handling systems as a primary example. We report on a feasibility study conducted for the material handling systems of a major European airport. The results show that the use of the performance spectrum enables much better predictions than baseline approaches.**

## I. Introduction

*Predictive Process Monitoring* (PPM) is crucial for supporting the operation of *Material Handling System* (MHS), such as *Baggage Handling Systems* (BHS) of airports, where undesired or unexpected performance scenarios can lead to congestion, inefficient management of manual operations, baggage mishandling (e.g. being late for a flight) and as a result, to lower customer satisfaction and higher operational costs [1]. Predictive analysis of *Process Performance Indicators* (PPI), which can reveal such a problematic scenario in advance allowing to take mitigating actions. While classically the problem is addressed through manually built simulation models [2], [3], also event data generated by material handling processes are available describing movement of materials or manual operations in the past. Current Machine Learning (ML) approaches for PPM in MHS [4] make limited use of the process dimension in the data. In recent years a variety of PPM approaches have appeared for Business Processes (BP) [5]. However, these techniques mostly assume isolated cases and stationary processes, these assumptions are violated for MHS [6]. Recently proposed PPM approaches for business processes [5] focus on predicting a single case assuming isolated cases and stationary processes. On one hand, these assumptions are violated for MHS and most business processes [6]. The performance of each case is dependent on the performance of the cases "around", i.e., the recent state of the system itself and the recent performance of handling groups of cases, rather than on an individual case performance or properties. In such cases, for the instance inter-case similarity-based features are required for predicting remaining time until case completion [7]. On the other hand,

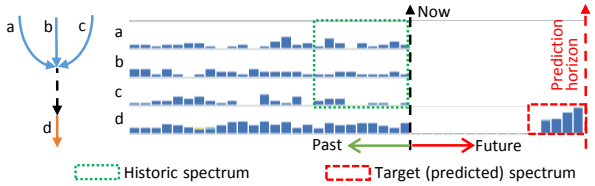



Fig. 1. The load peak on the X-ray baggage screening machine of a BHS (process step d) can be computed through the recent historic load on the check-in counters (process steps a-c) rather than individual properties of cases.

relevant PPIs in practice [5], including MHS, are not measures for an individual case but *aggregate* measures such as the amount of work (cases) expected or the occurrence of high waiting times (for all cases) at a particular step in a specific time-window from now. In this paper, we consider the problem of predicting aggregate PPM in (non-stationary) processes competing for shared resources, e.g., baggage handling in an MHS under changing load. In the following we refer to this as the *Inter-Case Performance Prediction Problem* (IC3P). The *Performance Spectrum* (PS) [6] formats performance data from process event logs in a way that maps the performance of each case over each process step over time. The PS reveals the performance of all cases in a step in relation to preceding and succeeding steps, non-stationarity of performance, and mutual influences of cases over time in detailed and aggregated form. In the following, we show that aggregate PPM can be predicted directly from features of the PS as the richness of the PS directly encodes inter-case performance characteristics over time. Fig. 1 illustrates a simplified PS that shows that the workload and performance of all cases at step d in the future depends on the recent throughput in steps a-c *together*. This allows us to reduce the IC3P problem to an ML problem over features of the PS. To achieve this result, we introduce the notion of "feature channels" to capture different process and performance perspectives in the PS and to describe different PPIs over time and over process steps.

We provide a general formal formulation of the inter-case performance prediction problem over the multi-channel PS; and a methodology for formulating problem instances (especially feature selection and reduction) and solving the problem (using standard ML for model training). Our evaluation on simulated and real-life data demonstrates the feasibility of our approach and that prediction of aggregated PPM using PS-based features

978-1-7281-0919-0/19/$31.00 ©2019 IEEE
DOI 10.1109/ICPM.2019.00029

outperforms prediction using case similarity-based features [7].

The remainder of this paper is structured as follows. We discussed work related to PPM of MHS and BP in Sect. II. We recall the PS and introduce the multi-channel PS in Sect. III. We formally define the generic problem of the IC3P in Sect. IV and propose the methodology for solving it in Sect. V. We report on our evaluation on synthetic and real-life event logs in Sect. VI and discuss our findings and future work in Sect. VII.

## II. Related Work

For BP, the remaining processing time for a case can be predicted by regression models [8] or by decorating a transition system with remaining time [9], prior trace clustering improves the prediction [10]. In [11], a Naive Bayes classifier predicts the future path of a single running case and a regression model predicts the transition durations on this path. The likelihood of future activities can be predicted using Markovian models [12], but without providing any time predictions. Completion time of the next activity can be predicted by training an LSTM neural network [13], or by learning process models with arbitrary probability density functions for time delays through non-parametric regression from event logs [14] that can also be used for learning simulation models to predict performance [15], [16]. Competing for shared resources can be taken into account through simulation models or with queuing models [17]. Using only features of a single case, these models cannot predict PPIs for non-isolated cases. Estimating an aggregate PPI through the outcome of individual cases [18] cannot be used for non-mandatory outcome of non-isolated cases. Prediction of the remaining time for a single case in processes with non-isolated cases is addressed in [7], intra-case features of a running case of interest are coupled with inter-case features of concurrently running cases, "close" to the case of interest in terms of control-flow and temporal distances. However, in processes with tightly coupled dynamics such as MHS, cases influence each other, e.g., congestions propagate through the system and resource problems affect groups of cases, impacting the performance. The PS-based approach in this paper specifically captures this dynamics. Since [7] is the current state-of-the-art approach for inter-case feature encoding, we use it as a baseline.

Among MHS, BHS are studied extensively. In the BHS domain, relationships between some bag- and system-related properties can be learned by feedforward NN models [4], but the results reported as just acceptable, even for a fully controllable environment of a simulation model. A risk of baggage mishandling can be predicted with an aggregated probabilistic flow graph as a function of travel durations between system locations [1], while dynamic routing is not supported. Problem-oriented *simulation models* allow identifying of bottlenecks and critical operations for inbound baggage handling [19]; learning dependencies between security policies and time characteristics of manual baggage screening [3]. In [2] an overview of various simulation-based performance prediction techniques for baggage screening is provided. While these simulation models are precise, their design requires in-depth knowledge of a system design and proved to be time-consuming.

Our work contributes to the problem of predicting aggregate PPIs for processes with *non-isolated* cases that *influence each other*. We capture inter-case dependencies by leveraging the performance spectrum that we recall next, and learn unknown system behavior from performance-related features of the performance spectrum, thereby extending the application of non-simulation-based approaches of PPM to MHS.

## III. Performance Spectrum

We first establish some basic notations for events and logs, recall the idea of the Performance Spectrum (PS) and revise the definitions of [6] to provide "elementary" PS building blocks from which we construct a *multi-channel* PS for performance prediction.

Let $A$ be a set of *event classifiers*; $A$ is usually the set of activity names, but it may also be the set of resource names, or locations. Let $T$ be the set of time durations and time stamps, e.g., the rational or real numbers. Let $\mathcal{E}$ be the *universe of events* with *attributes*, and let $AN$ be a set of attribute names. For any $e \in \mathcal{E}, n \in AN$, $\#_n(e)$ is the value of attribute $n$ for event $e$ ($\#_n(e) = \perp$ if attribute $n$ is undefined for $e$). Each event has mandatory attributes *time*, $\#_{time}(e) \in T$ and *act*, $\#_{act}(e) \in A$. As a short-hand, we write $e^{(a,t)}$ to indicate that event $e$ has $\#_{act}(e) = a$ and $\#_{time}(e) = t$. Let $\mathcal{Z}$ be the *universe of cases* with attributes. For any $z \in \mathcal{Z}, n \in AN$, $\#_n(z)$ is the value of attribute $n$ for case $z$. Each case has a mandatory attribute *trace*, $\#_{trace}(z)$, defining a finite sequence of events $\#_{trace}(z) = \sigma \in \mathcal{E}^*$. For $\sigma = \langle e_1, \ldots, e_n \rangle$, we write $|\sigma| = n$ and $\sigma_i = e_i, i = 1, \ldots, n$. An event log is a set of cases $L \subseteq \mathcal{Z}$ where no two traces share an event.

The *Performance Spectrum* is a *data structure* introduced in [6] to describe the performance of process steps over time. We first recall the idea and then adopt it for performance prediction. We call $(a, b) \in A \times A$ a *process segment* describing a step from activity $a$ to activity $b$, hand-over of work from resource $a$ to $b$ or the movement of goods from location $a$ to $b$. Each occurrence of a segment $(a, b)$ in a trace $\langle \ldots, e_i^{(a,t_a)}, e_{i+1}^{(b,t_b)}, \ldots \rangle$ allows to measure the time $t_b - t_a$ between occurrences of a and b. A histogram $H = H(a, b, L) \in \mathbb{B}(T)$ describes how often all the *time differences* $t_b - t_a$ between $a$ and $b$ have been observed in $L$. In contrast, the *performance spectrum* $\mathbb{S}(a, b, L)$ collects the actual *time intervals* $(t_a, t_b)$ observed in $L$. Fig. 2(a) shows the so-called *detailed PS* for the segment $(a, b)$: each dot along the $a$-axis marks an occurrence of an $a$-event, correspondingly $b$-events are shown on the $b$-axis. The diagonal line $(a_1, b_1)$ describes one *occurrence* of the segment from event $e_i^{(a,t_a)}$ to $e_{i+1}^{(b,t_b)}$ in the same trace, e.g., the movement of a bag between two locations. Different lengths of segment occurrences indicate performance differences for different cases; changing density of segment occurrences indicates changing workload on the segment over time.

To allow *computing* with the visually evident performance-related features of Fig. 2(a), the PS may also provide *classification* and *aggregation* of the occurrences of a segment. In the following, we revise these definitions to allow defining basic building blocks for process segment, classification, and time

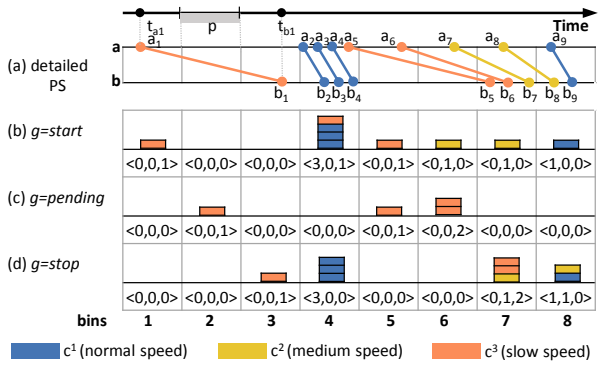



Fig. 2. In the detailed PS (a) the color-coded lines show cases with different speed classes, while the aggregated PS with various grouping (b-d) capture various performance aspects of case handling for each time window.

interval of interest. In [6], occurrences $\langle \dots, e_i^{(a,t_a)}, e_{i+1}^{(b,t_b)}, \dots \rangle$ of $(a,b)$ were classified wrt. duration, e.g., the actual duration $\Delta t = t_b - t_a$ or whether $\Delta t$ is in the 25% quartile of the histogram $H(a,b,L)$. Here, to enable general performance prediction, we assume a function $\mathbb{C}$ that maps two events $e_i, e_{i+1}$ to a *performance class* $\mathbb{C}(e_i, e_{i+1}, L) = c \in C$ considering *any* properties of $e_i, e_{i+1}$ and the log $L$ in which they occur. We call $\mathbb{C} : \mathcal{E} \times \mathcal{E} \times 2^{\mathcal{Z}} \to C$ a *performance classifier* for $C$. Examples are the duration from $e_i^{(a,t_a)}$ to $e_{i+1}^{(b,t_b)}$, remaining time until case completion since $e_{i+1}$, or whether a material had to be routed from $a$ to $b$ because an alternative route from $a$ was blocked; scalar values may be abstracted to categories.

If the performance classes $C$ are *finite*, the detailed PS of a segment $(a,b)$ can be *aggregated* over "bins" of a chosen, fixed duration $p$, called *bin size*. For each bin $b_j$ and class $c \in C$ we count how many occurrences of segment $(a,b)$ of class $c$ occur "during" $b_j$. As Fig. 2 illustrates, we may choose to count the segments that *start* during $b_j$ ($e_i$ is in $b_j$ but $e_{i+1}$ is not) (b), *stop* during $b_j$ (d) or are *pending* (c). For example, segment occurrence $a_1 b_1$ has class $c^3$ (color pink), starts in bin 1, is pending in bin 2 and ends in bin 3. Suppose we chose to group on *start*, then, we aggregate this information into a vector $\langle v_j^1, v_j^2, v_j^3 \rangle$ where, say, $v_j^3$ counts the number of segment occurrences of class $c^3$ that occurred during bin $j$. Figure 3(b-d) shows the aggregation vectors for each grouping and each bin and their visualization as stacked barcharts. Def. 1 and 2 formalizes these for a single segment, classifier, and bin. We canonically lift them to multiple classes, segments, and bins afterward.

**Definition 1** (Detailed performance spectrum (Detailed PS))**.** *Let $\mathbb{C}$ be a performance classifier for $C$. Let $L$ be a log and $(a,b)$ be a segment. The* detailed PS *of $(a,b)$ in $L$ wrt. $\mathbb{C}$ is the bag* $\mathbb{S}_L((a,b),\mathbb{C}) = [(t_a, t_b, c) \mid \langle \dots e_i^{(a,t_a)}, e_{i+1}^{(b,t_b)} \dots \rangle = \#_{trace}(z), z \in L, 1 \le i < |\#_{trace}(z)|, c = \mathbb{C}(e_i^{(a,t_a)}, e_{i+1}^{(b,t_b)}, L)] \in \mathbb{B}(T \times T \times C)$.

In Fig. 2(a) elements $(t_a, t_b, c)$ of the PS are visualized as lines that start at time moments $t_a, t_b$ on axes a,b, class c is indicated by color.

**Definition 2** (Aggregation of PS)**.** *Let $\mathbb{C}$ be a performance classifier for finite classes $C = \{c_1, \dots, c_k\}$. Let $S = \mathbb{S}_L((a,b),\mathbb{C})$ be the detailed PS of a segment $(a,b)$ in a log $L$. Let $p \in T$ be a duration we call the* bin size *and let $g \in \{start, stop, pending\}$. The occurrences of $(a,b)$ in bin $j \in \mathbb{N}$ (of length $p$) regarding grouping $g$ is the multiset $b_j$ such that:*

- $b_j = [(t_a, t_b, c) \in S \mid j \cdot p \le t_a < (j+1) \cdot p]$ *if $g = start$,*
- $b_j = [(t_a, t_b, c) \in S \mid j \cdot p \le t_b < (j+1) \cdot p]$ *if $g = stop$, and*
- $b_j = [(t_a, t_b, c) \in S \mid j \cdot p > t_a \wedge t_b \ge (j+1) \cdot p]$ *if $g = pending$ (the segment starts before the start of the bin, and ends after (or at) the end of the bin).*

*The* aggregation *of $S$ over bin $j$ and grouping $g$ is the vector $v_j = \langle v_j^1, \dots, v_j^k \rangle \in \mathbb{N}^k$ counting how often performance class $c^i$ occurred in bin $v_j$: $v_j^i = |\{(t_a, t_b, c^i) \mid (t_a, t_b, c^i) \in b_j\}|$. Let $\mathbb{S}_L((a,b), \mathbb{C}, g, p, j) = v_j$.*

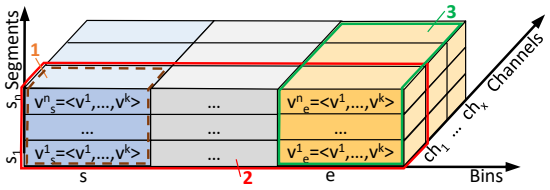



Fig. 3. The multi-channel Performance Spectrum.

For example, in Fig. 2(d) the aggregation of the PS for segment $(a,b)$ over bin 7 and grouping $g = stop$ is vector $\langle 0,1,2 \rangle$ which counts ends of all segment occurrences in this bin: zero for class $c^1$, one for $c^2$ ($b_7$) and two for $c^3$ (points $b_{5-6}$).

The aggregation of a detailed PS into a bin has 3 main dimensions: (1) the segment $(a,b)$, (2) the parameters describing the bins, i.e., the classification $\mathbb{C}$, the grouping $g$, and the period $p$, and (3) the bin number $j$. To simplify notation, we call the bin parameters $ch = (\mathbb{C}, g, p)$ a *PS channel*, and write $\mathbb{S}_L((a,b), ch, j) = v_j$ for the aggregation vector of Def. 2.

We now show how a bin $\mathbb{S}_L((a,b), ch, j)$ of the aggregated PS is the basic building block to formulate various performance prediction problems. Consider Fig. 3: each bin of the aggregated PS can be placed in a 3-dimensional space defined by a series of segments $SEG = \langle (a_1, b_1), \dots, (a_n, b_n) \rangle$, a series of channels $CH = \langle ch_1, \dots ch_x \rangle$, and a time interval of bins $[s,e] = \langle s, s+1, \dots, e \rangle$ of interest. We use slicing and dicing in this 3d-data structure to define our prediction tasks. Adopting notation from algebra software, we let the arguments of $\mathbb{S}_L(\cdot,\cdot,\cdot)$ range over sequences of segments, channels, and bin numbers to denote rows, columns, matrices, and cubes of bins along those dimensions. Let $SEG = \langle (a_1, b_1), \dots, (a_n, b_n) \rangle$ be a sequence of segments, $CH = \langle ch_1, \dots, ch_x \rangle$ be a sequence of PS channels (of identical period $p$), $ch \in CH$, and $[s,e] = \langle s, s+1, \dots, e \rangle$ a sequence of bin numbers, $j \in [s,e]$. We write $\mathbb{S}_L(SEG, ch, j)$ for the column vector $\langle \mathbb{S}_L((a_1,b_1), ch, j), \dots, \mathbb{S}_L((a_n,b_n), ch, j) \rangle^\top$. Note that this vector consists of vectors $v_j^i$ for each segment $(a_i, b_i)$ and bin $j$. In Fig. 3 such a column vector corresponds to a column of blocks of size $n \times 1 \times 1$, e.g. area (1).

We write $\mathbb{S}_L(SEG, ch, [s,e])$ for the row vector $\langle \mathbb{S}_L(SEG, ch, s), \dots, \mathbb{S}_L(SEG, ch, e) \rangle$. Note that each $j^{th}$ entry of this vector corresponds to a column vector $\mathbb{S}_L(SEG, ch, j)$. In Fig. 3 $\mathbb{S}_L(SEG, ch, [s,e])$ corresponds to a

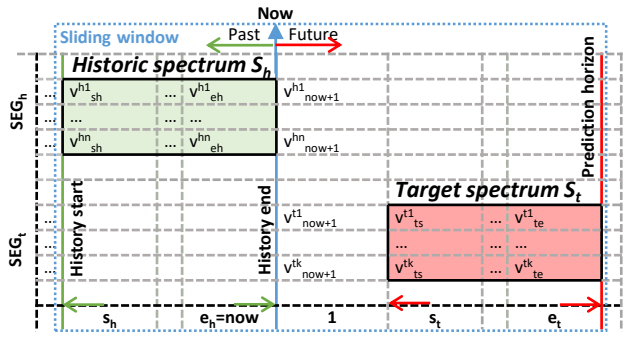



Fig. 4. The configuration of the historical and target spectra "around" the current time of the sliding window: the historic spectrum in the past is used to compute the target spectrum in the future.

frontal 'slice' of size $n \times 1 \times (e - s + 1)$ for channel $ch$ , e.g. area (2). We use this matrix to *visualize* the performance spectrum of the segments $SEG$ over the time period $[s, e]$ in a single channel $ch$. For example, Fig. 2(b) visualizes one row of such a matrix and Fig. 3 visualizes an entire matrix. We also this visualization of the frontal slice for feature selection in Sect. V. Performance prediction requires considering information from *multiple* channels during the same time period. We write $\mathbb{S}_L(SEG, CH, j)$ for the column vector $\langle \mathbb{S}_L(SEG, ch_1, j), \ldots, \mathbb{S}_L(SEG, ch_x, j)\rangle^\top$. Note that each $r^{th}$ entry of this vector corresponds to a column vector $\mathbb{S}_L(SEG, ch_r, j)$. In Fig. 3 $\mathbb{S}_L(SEG, CH, j)$ corresponds to a vertical 'slice' of a single bin columns of size $n \times x \times 1$, e.g. area (3), where it is shown as a matrix over segments and channels. Note that the order of segments and channels is arbitrary but fixed to allow implicit encoding of features, whereas the order of bins is determined by time.

We write $\mathbb{S}_L(SEG, CH, [s, e])$ for the row vector $\langle \mathbb{S}_L(SEG, CH, s), \ldots, \mathbb{S}_L(SEG, CH, e)\rangle$, which correponds to the whole cube in Fig. 3. Note that this vector is a matrix with columns corresponding to bins, i.e. time, and rows corresponding to segments and channels. Such a structure allows to slice and dice it in various ways. Row vectors can be used for visual analytics, columns vectors of various bin intervals can serve for extracting independent and dependent variables for model training. Aggregation along the bin and segment axes allows for feature space reduction. We call $\mathbb{S}_L(SEG, CH, [s, e])$ the *multi-channel* performance spectrum of $L$ over segments $SEG$, channels $CH$, and period $[s, e]$.

## IV. Problem Statement

The Inter-Case Performance Prediction Problem (IC3P) is to obtain a model for predicting the performance of multiple cases together, in a specific part of the process, within a particular prediction window. In this section, we show how the features of the multi-channel PS of Sect. III allow formulating precise IC3P instances.

### A. Problem Formulation with Performance Spectrum

Along the dimensions of the multi-channel PS, the IC3P is to predict the performance characteristics for segments of interest for a time-interval in the future (the *target* spectrum), based on the performance of relevant segments during a recent time interval (the *historic* spectrum). An estimate for a specific PPI can then be derived by aggregating performance-related features of the target spectrum. In Fig. 4 a schematic configuration of such a problem is shown for one channel of a multi-channel PS. The historic spectrum is specified by a sequence $SEG_h = \langle s_h^1, \ldots, s_h^n\rangle$ of segments and a bin interval $[s_h, e_h]$ where $s_h < e_h \leq 0$ define *offsets* from the current bin *now* in the sliding window. The target spectrum is given by segments $SEG_t$ and a bin interval $[s_t, e_t], 0 \geq s_t > e_t$ with offsets into the future. Index $e_t$ defines the prediction horizon (PH). Both historic and target spectrum are defined over the same sequence $CH$ of channels. This allows formulating IC3P as a regression problem, using historic and target spectra as a source of independent and dependent variables over common time parameter $T$, formalized in (1):

$$\begin{aligned} &\mathbb{S}_L(SEG_t, CH, [s_t + T, e_t + T]) = \\ &f(\mathbb{S}_L(SEG_h, CH, [s_h + T, e_h + T])) + R, \end{aligned} \tag{1}$$

or $S_{L,t}(T) = f(S_{L,h}(T)) + R$ for short. Function $f$ *predicts* values of the target spectrum, and $R$ is a residual, i.e. the deviation between observed and predicted values. To learn $f$, we use the sliding window method [20] for selecting $w$ samples $(S_{L,h}(T_i), S_{L,t}(T_i))$ of historic and target spectrum for times $T_1, \ldots, T_w$, and apply a ML method to learn $f$ from these samples. By comparing the actual values $y = \langle S_{L,t}(T_1), \ldots, S_{L,t}(T_w)\rangle$ in the target spectrum with the values predicted by learned function $f$, $y' = \langle f(S_{L,h}(T_1)), \ldots, f(S_{L,h}(T_w))\rangle$, we can estimate the prediction error $R$ in $f$ by a function $error(y, y') \in \mathbb{R}$. In general, a target spectrum does not contain the target PPI directly, but contains performance-related features sufficient to compute it. For that, we define the following function:

$$ppi(T) = g(\mathbb{S}_L(SEG, CH, [s_t + T, e_t + T])) + \varepsilon, \tag{2}$$

where error $\varepsilon = ppi(T) - ppi'(T)$ and $ppi'(T)$ is the predicted target PPI observed over interval $[s_t, e_t]$.

### B. Examples of Problem Instances

We now instantiate the generic problem formulation from (1) for concrete real-life performance prediction problems of a major European airport baggage handling system (BHS). A fragment of its simplified material flow diagram is shown in Fig. 5. We first consider the process from check-in until screening. Bags enter the system via one of several dozen check-in counters $a_1^1$-$a_n^m$ and then move via conveyor belts to one of two pre-sorter loops P1,P2 where each bag has to go to the X-ray baggage screening machines, e.g. entering via $(E_1, S_1)$ and leaving via $(S_2, X_1)$. For operational support, the main concern is to keep the BHS performance steady at some desired level. In particular, the workload in a processing step or system part may not exceed its capacity, as this otherwise leads to long queues or stalling of sorting loops. Here, workload *prediction* is central for proactive management. One concrete problem (*PI1*) is to predict the load (in bags per minute) at the X-ray baggage screening machines (SM) on $t_{PH} = 4$ minutes

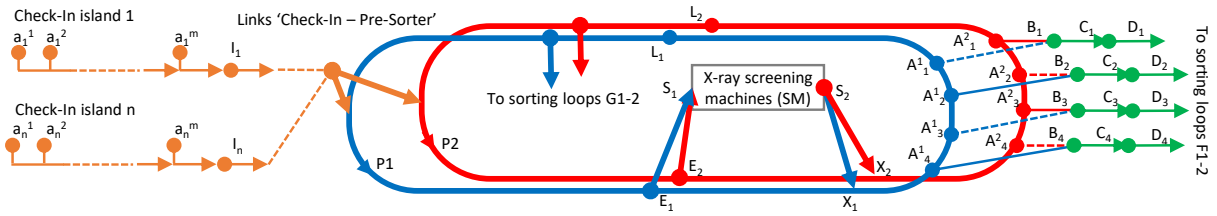



Fig. 5. Check-in and pre-sorting areas of a Baggage Handling System.

in advance for P1. In Fig. 5 this load corresponds to the load on segment $SEG_{t,PI1} = \langle (E_1, S_1) \rangle$.

To express this problem in terms of (1), we define the target and historic spectra. First, to represent the load, we define a PS channel with a single class (load does not distinguish different classes), grouping *start* and 1-minute bin: $ch_{PI1} = \langle (\mathbb{C}_{PI1}, start, 1) \rangle$, where $\mathbb{C}_{PI1}$ returns zero for any segment occurrence. For log $L$, the target spectrum for *PI1* is $S_{PI1}(T) = \mathbb{S}_L(SEG_{t,PI1}, ch_{PI1}, [t_{PH} + T, t_{PH} + T])$. The predicted load is the sum of all its values, in bags per minute. Then we make the following hypothesis: the load depends on the average load of the check-in counters in 1-3 minutes before *now*. To capture that, we include the check-in segments to the historic spectrum: $SEG_{h,PI1} = \langle (a_1^1, I_1), \ldots, (a_1^m, I_1), \ldots, (a_n^1, I_n), \ldots, (a_n^m, I_n) \rangle$ and time-interval $[s_h^{PI1}, e_h^{PI1}]$ as $[-3, -1]$. This leads to the following regression problem derived from (1):

$$S_{PI1}(T) = f_{PI1}(\mathbb{S}_L(SEG_{h,PI1}, ch_{PI1}, [T-3, T-1]) + R. \quad (3)$$

The target PPI is defined as $ppi_{PI1}(T) = S_{PI1}(T)_1$, where the index means the aggregate of the fist performance class in $\mathbb{C}_{PI1}$.

Another concern of BHS operational support is predicting the risk that baggage being late for a flight; we now instantiate (1) for this problem (*PI2*). The second part of the process in Fig. 5 moves bags from the screening machines to sorting loops F1,F2 (exit the pre-sorter P1 and P2 via $A_1^1$-$A_2^4$, $B_1^1$-$B_2^4$). It may happen that, for instance, a bag on P1 that has to go to F1, cannot be diverted onto any of the conveyors $(A_i^1, B_i)$ because these are unavailable (e.g., due to high load on all $(C_i, D_i)$. In this case, the bag will be looping on P1 until it can be diverted successfully. Each round increases the bag's *estimated time to destination* (EST) $t_{est}$ by the loop duration $t_P$. If the new estimate $t'_{est} = t_{est} + t_p$ exceeds the deadline when the bag has to arrive at its destination to reach the flight, the bag is expected to be late and correcting actions, e.g. making the bag priority higher, can be undertaken.

So, to predict such late bags, it is sufficient to predict extra re-circulation due to unavailability of diverts $A_1^1$-$A_2^4$. We formulate *PI2* as a problem of predicting such re-circulation for P1. On P1, any bag traveling the segment $(A_1^1, L_1)$ is re-circulating (as it could not be diverted to F1,F2), thus the segments of the target spectrum are $SEG_t = \langle (A_1^1, L_1) \rangle$. Selecting $t_{PH} = 60seconds$, duration-based classifier $\mathbb{C}_{PI2}$ (whether $t = t_b - t_a$ is in the 25%-quartile of the histogram $H(a, b, L)$) and $chs_{PI2} = \langle (\mathbb{C}_{PI1}, start, 30seconds), (\mathbb{C}_{PI2}, pending, 30seconds) \rangle$, we make a hypothesis, that the target spectrum depends on the load and delays of $SEG_{h,PI2} = \langle (S_2, X_1), (S_2, X_2), (A_i^1, B_i), (A_i^2, B_i), (B_i, C_i), (C_i, D_i) \mid i = 1, \ldots, 4 \rangle$ for two bins before *now*. We predict the target spectrum $S_{PI2}(T) = \mathbb{S}_L(SEG_{t,PI2}, chs_{PI2}, [T+1, T+1])$ as follows:

$$S_{PI2}(T) = f_{PI2}(\mathbb{S}_L(SEG_{h,PI2}, chs_{PI2}, [T-2, T-1]) + R. \quad (4)$$

The PPI is defined as $ppi_{PI2}(T) = \mathbb{S}_L(SEG_{t,PI2}, chs_1, [T+1, T+1])_1 + \varepsilon$, i.e. we select channel with grouping *start* and the first performance class in $\mathbb{C}_{PI2}$.

## V. Approach

In Sect. IV, we showed that prediction of aggregate performance measures for non-isolated cased can be expressed as a generic regression problem over the performance spectrum. In this section, we present a general methodology on formulating concrete problem instances and how to solve them using a standard machine-learning pipeline. Fig. 6 illustrates the overall methodology.

The main challenge is to correctly select the features for defining the historic and the target spectra. In the following, we summarize some lessons learned from our experiments that we discuss in Sect. VI.

### A. Methodology

In Step 1, *target segments* for the problem, i.e. the segments, which performance-related features are sufficient for computing the target PPI, are identified and located in the model. In Step 2, the target segments are considered for aggregation. MHS equipment is usually redundant, to provide high availability and fault tolerance of the whole system. For example, several baggage screening machines, working in parallel, are usually grouped in a cluster with symmetrical layout and some load balancing policy. A set of similar-purpose segments $(a_1, b_1), \ldots, (a_n, b_n)$ can be aggregated into a new aggregated segment $(a^*, b^*)$ by relabeling $a_i \mapsto a^*$, $b_i \mapsto b^*$ prior to computing the PS.

Similar-purpose segments within such clusters can be aggregated in the PS to reduce the feature vector along the *SEG* dimension (see Fig. 3). During Step 3 we define PS channels that contain features, required for computing the target PPI, by choosing a common granularity (period $p$), classifiers and groupings. This step is specific to the problem. For example, to compute load on a segment, a combination of grouping *start* with a constant (single-value) classifier may be sufficient, as for *PI1*, while for counting performance outliers

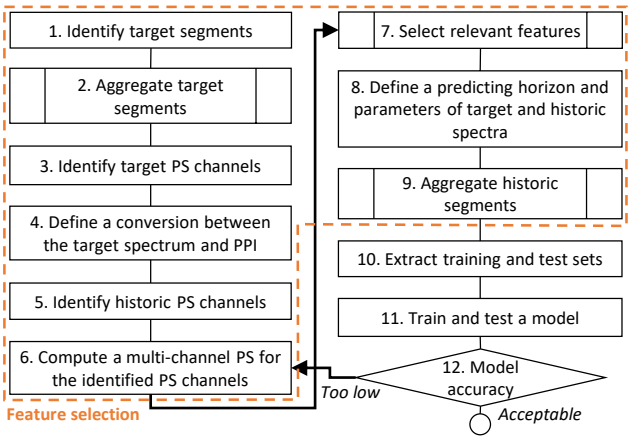



Fig. 6. The methodology on formulating problem instances of the inter-case performance prediction problem.

another grouping *pending* with a duration-based classifier is required. Then in Step 4, given the identified target spectrum parameters, a concrete function $g$ (2) is defined. During Step 5, historic PS channels are identified to take into account more features for estimating the target spectrum. While period $p$ is common for all the PS channels, particular classifiers and groupings for these channels depend on the problem and system. Using domain knowledge and/or performance analysis results of earlier iterations, additional PS channels can be included into the historic PS channels vector. Next in Step 6, a multi-channel PS is computed for the identified PS channels.

In Step 7 we should answer the following question: which features of the multi-channel PS influence the target spectrum and should be included in the historic spectrum? For that historic segments and time boundaries of the historic spectrum should be defined. We suggest using the computed multi-channel PS as a visual analytics technique for feature selection according to the following guideline. We formulate the following high-level guideline, which describes the main steps of such analysis. First, a *Segment Group of Candidates* (SGC) to the historic spectrum is identified. The focus is usually on segments that are in several steps upstream and downstream the target segments. Additionally, all segments that *a priori* affect the target spectrum (according to domain knowledge) are included. Afterward, the correlation between the target spectrum features and features of segments in the SGC can be determined. For example, in Fig. 7 an interval of higher load on segment $s_1$ causes a higher load on target segment $s_t$, so this segment should be included into the historic spectrum. Finally, the following questions should be answered. Which segments of the SGC influence the target spectrum? What is an average delay of affecting the target spectrum ($\Delta_1$ in Fig. 7)? What is the time interval of the historic spectrum that should be used to estimate the target spectrum ($l_1$ in Fig. 7)?

In Step 8, the Prediction Horizon (PH), i.e., the moment of prediction, is chosen in light of the dynamics from the segments and bins in the historic spectrum that dominate the target spectrum. After this step completion, all required parameters

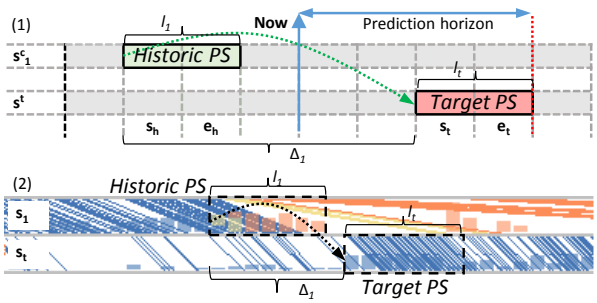



Fig. 7. Visual analytics over the PS: segment $s_1$ influences target segment $s_t$.

of the target and historic spectra are defined: the segments names, start and stop indices. Similarly to Step 2, in Step 9 the historic segments are considered for aggregation.

In Steps 10-11 a standard ML pipeline is exploited for model training. The multi-channel PS, built in Step 6, is used directly for the extraction of the training and test sets. Using the sliding window technique [20], the historic and target sets are instantiated for each bin of the multi-channel PS, using the parameters identified in the previous steps, and stored as a sample of the training or test set for the consecutive model training. After a model is configured, trained and tested, a decision on the model accuracy is made. If it is lower than required, more iterations can be done, e.g. to change the non-target PS channels, aggregation rules, selected features, PH and model configuration in order to improve the model accuracy.

In Sect. VI we will apply this approach on *PI1* and *PI2*.

## VI. Evaluation

We extended the interactive ProM plug-in 'Performance Spectrum Miner' with the multi-channel PS and scripts for training models in the PyTorch ML framework[1]. We demonstrated the feasibility of our approach and compared it to the current state-of-the-art approach [7] by training models for *PI1* and *PI2* of Sect. IV-B (details of data are in Sect. VI-A and VI-B). For the experiments we applied the training-validation-test approach, using 20% of the data for testing. The remaining part was randomly shuffled and used for model training and 5-fold cross-validation in proportion 4:1. For model learning, we evaluated three approaches. (1) For our approach, we extracted PS-based features as discussed in Sect. IV and trained Logistic Regression (LR) and Feedforward (FF) Neural Network (NN) models that predict the expected load in the target segments directly. (2) As a baseline capturing both intra- and inter-case dependencies, we chose [7][1]. As it only predicts PPIs for individual cases, we had to adopt it for the aggregate PPI as follows. First, we trained a model for predicting the time between *last events of trace prefixes* that end with occurrences of historic spectrum segments and *starts of target segments*, using LR and FF NN models. For each prefix the learned model predicts when it will reach the "target". By aggregating how many cases are predicted to reach the "target" bin, we can estimate the expected load. Because such an aggregation can be done only for cases that eventually

[1]the simulation event log, ProM plugin, PyTorch script and source code of [7] are available at https://github.com/processmining-in-logistics/psm/tree/ppm

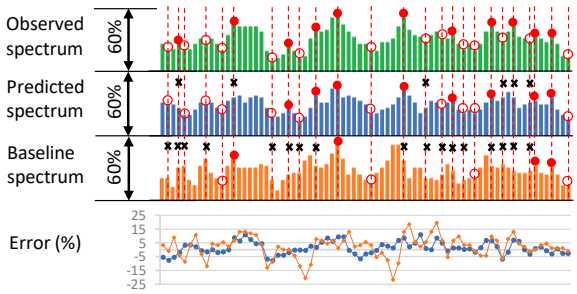



Fig. 8. The real, predicted (PS-based approach (1)) and baseline (approach (2)) load of the baggage screening machines, in % of the maximal load (top): each bin represents the load for one minute, filled and blank circles show matched peaks and dips, while the X's show mismatches. The residuals of the predicted load in % of max. load per bin (bottom): the baseline (orange) shows greater deviations than the PS-based model (blue).

reach target segments, it assumes beforehand knowledge about future paths of cases. This assumption holds for historic data on the model training stage, but does not hold for real MHS on the prediction stage. For example, in the real BHS, considered in Sect. VI-B, bags can be routed to P2 (see Fig. 5 and never reach the target segment $(E_1, S_1)$ (PI1), or bags from P1,P2 can be sent to sorters G1,G2 instead of F1,F2, thereby being outside the scope of the re-circulation problem (PI2). (3) As a naive baseline, we chose an average value of dependent variables, observed in a time interval $[s_h, e_h]$, corresponding to the historic spectrum. To measure errors (see Sect. IV), we computed *Root Mean Squared Error* (RMSE), *Mean Absolute Error* (MAE) and *R squared*, which is meaningful for linear models. Additionally, we did meticulous residual diagnostic of predictions for test sets. Models were trained on a server with 40 CPUs, six GPUs and 400 GB memory.

### A. Simulation Model of a Baggage Handling System

To generate an event log for addressing *PI1*, we designed a simulation model of a simple BHS, comprising a typical BHS layout: conveyors, a sorting loop, a baggage screening machine, divert and merge units. As a load, a check-in scenario with normally distributed distances between bags was replayed to generate an event log with 134.000 events and 11.518 cases for 84 operating hours. Events were recorded when bags passed through various locations in the system. The resulting training and test sets have 15 feature variables and 15.000 samples[1], on which we applied approaches (1-3). Table I (a) shows the resulting measures. Our PS-based models show two times smaller errors RMSE and MAE than approach (2). The PS-based LR model has a greater and closer to 1.0 R squared measure than the LR model of approach (2), i.e. it explains significantly more variable variations.

### B. Baggage Handling System of a Major European Airport

In this experiment, we addressed *PI1* and *PI2* for a Vanderlande-built BHS of a major European airport. In the event log, each case corresponds to one bag, events are recorded when bags pass sensors on conveyors, and activity names describe locations of sensors in the system. Events are recorded only when a bag is diverted to another conveyor, so information

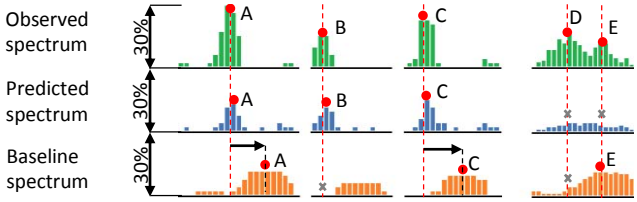



Fig. 9. The peaks of re-circulation within 30-second bins (in % of the max. load): the FF NN model predicted peaks A, B, C in the correct bins, whiles the baseline predicted them with the significant delay as a result of auto-correlation. Part of peaks (e.g. D and E) were not predicted by the model.

| Experiment | Approach | Model | R squared | RMSE | MAE |
|---|---|---|---|---|---|
| (a) BHS sim. model PI1 | (1) | LR | 0.82 | 12.1 | 8.2 |
| | (1) | FF NN | 0.84* | 11.6 | 7.7 |
| | (2) | LR | 0.35 | 23.2 | 15.3 |
| | (2) | FF NN | 0.46* | 21.2 | 16.2 |
| | (3) | - | - | 35.8 | 23.2 |
| (b) Real BHS PI1 | (1) | LR | 0.74 | 7.0 | 5.0 |
| | (1) | FF NN | 0.75* | 6.9 | 4.8 |
| | (2) | LR | 0.38 | 10.8 | 7.9 |
| | (2) | FF NN | 0.69* | 7.6 | 5.3 |
| | (3) | - | - | 11.0 | 7.9 |
| (c) Real BHS PI2 | (1) | LR | 0.05 | 3.0 | 1.8 |
| | (1) | FF NN | 0.45* | 2.4 | 1.3 |
| | (3) | - | - | 3.0 | 1.7 |

TABLE I
MODEL ERROR MEASURES FOR PS-BASED (1), [7]-BASED (2) AND NAIVE (3) APPROACHES. RMSE AND MAE ARE IN % OF MAX. LOAD (A,B) AND RE-CIRCULATION (C).
*R squared values for FF NN are provided for the sake of completeness.*

about the bag locations is significantly incomplete. For one day of operations, an event log contains on average 850 activities, 25.000-50.000 cases and 1-2 million events. The entire log contained 148 million events for 120 consecutive days. Events recorded in non-operating night hours were excluded from the log. For the test set, days of different months and days of week were selected.

First, we addressed *PI1*. Following the approach in Sect. V, for approach (1) we trained an LR model and a two-layer FF NN on a dataset with 68 features and 108.000 samples. As shown in Tab. I (b), both PS-based models have smaller RMSE and MAE errors than the baseline models, and the R squared measure of the LR model is also good. Fig. 8 shows how the LR model correctly predicts peaks and dips in the load of the scanning machines compared to the recorded data, suggesting that the model is adequate for the workload prediction. The LR model is preferred for the sake of simplicity.

Finally we addressed *PI2*. Again, for approach (1) we trained an LR model and a four-layer FF NN on a dataset with 148 features and 216.000 samples (two times more than for the previous experiment because of the shorter bin duration), using approach (3) as a baseline. Tab. I (c) shows almost zero R squared of the LR model, that indicates incapability to explain variable variations, i.e. the model cannot predict infrequent re-circulation peaks, while RMSE and MAE of FF NN models are smaller than corresponding values of the baseline. Fig. 9 shows that the FF NN model correctly predicts moments of peaks in re-circulation, but consistently underestimates its actual amount, while the baseline demonstrates auto-correlation.

Despite incompleteness of the log, the sound PS-based

models, trained during the experiments, demonstrated the feasibility of the suggested approach for PPM problems of the real BHS as well as for the simulation model.

## VII. Conclusion

In this paper, we studied the problem of forecasting performance of non-stationary processes where cases influence each other through shared resources. We showed that the performance spectrum (PS) [6] derived from the event log of a process allows to model a variety of process performance features over time, capturing also inter-case dependencies. Specifically, we provided a basic building block defined over the three basic dimensions of process step, performance measure, and time interval. We showed that combining multiple such blocks of features in a *multi-channel* PS along the three dimensions allows formulating a large class of performance prediction problems as a regression problem. We proposed a methodology of solving this problem as a ML task, using the historical and target spectrum features as independent and dependent variables. The methodology includes the approach for feature selection, based on visual analytics of individual channels within the multi-channel PS, and process model-based aggregation of process segments for feature dimensionality reduction. We provided examples of real-life problem instances for a BHS and evaluation of our approach by training sound models for solving these problem instances on the real event log of a major European airport BHS. We demonstrated feasibility of our approach and compared it to the current state-of-the-art approaches, e.g., [7]. The experiments showed that our PS-based linear model outperforms more complex NN model of [7]-based approach, and the PS-based NN model outperforms the naive baseline for the problem instance where [7] is not applicable due to the optionality of the target process step. This work has several limitations. First, although supported by a methodology, feature selection and reduction requires domain knowledge and expertise. We expect that including a formal model of the process may help in engineering features from the performance spectrum. Second, while our models are technically sound, they still require validation in practice; we expect the need for higher accuracy, especially for predictions requiring a longer prediction horizon. Finally, the limitation of this work is that we only demonstrated the feasibility on MHS. Although the approach itself is generic and can be applied to event logs from various domains besides MHS, it does not take into account intra-case features of individual cases that are crucial for PPM of business processes. For adopting our approach for business processes, we aim to combine features of both aggregate PS-based and case-based PPM in future work.

## Acknowledgment

The research leading to these results has received funding from Vanderlande in the project “Process Mining in Logistics”.

## References

[1] T. Ahmed, T. B. Pedersen, T. Calders, and H. Lu, “Online risk prediction for indoor moving objects,” in *2016 17th IEEE International Conference on Mobile Data Management (MDM)*, vol. 1, June 2016, pp. 102–111.

[2] J. Skorupski, P. Uchroński, and A. Łach, “A method of hold baggage security screening system throughput analysis with an application for a medium-sized airport,” *Transportation Research Part C: Emerging Technologies*, vol. 88, pp. 52 – 73, 2018.

[3] S. Nahavandi, B. Gunn, M. Johnstone, and D. Creighton, “Modelling and simulation of large and complex systems for airport baggage handling security,” in *Intelligent Computing*, K. Arai, S. Kapoor, and R. Bhatia, Eds. Cham: Springer International Publishing, 2019, pp. 1055–1067.

[4] A. Khosravi, S. Nahavandi, and D. Creighton, “Estimating performance indexes of a baggage handling system using metamodels,” *Proceedings of the IEEE International Conference on Industrial Technology*, pp. 1 – 6, 03 2009.

[5] A. E. Márquez-Chamorro, M. Resinas, and A. Ruiz-Cortés, “Predictive monitoring of business processes: A survey,” *IEEE Transactions on Services Computing*, vol. 11, no. 6, pp. 962–977, Nov 2018.

[6] V. Denisov, D. Fahland, and W. M. P. van der Aalst, “Unbiased, fine-grained description of processes performance from event data,” in *Business Process Management*, M. Weske, M. Montali, I. Weber, and J. vom Brocke, Eds. Cham: Springer International Publishing, 2018, pp. 139–157.

[7] A.Senderovich, C.D.Francescomarino, and F.M.Maggi, “From knowledge-driven to data-driven inter-case feature encoding in predictive process monitoring,” *Information Systems*, 2019. [Online]. Available: https://doi.org/10.1016/j.is.2019.01.007

[8] B. F. van Dongen, R. A. Crooy, and W. M. P. van der Aalst, “Cycle time prediction: When will this case finally be finished?” in *OTM Conferences*, 2008.

[9] W. M. P. van der Aalst, H. Schonenberg, and M. Song, “Time prediction based on process mining,” *Inf. Syst.*, vol. 36, pp. 450–475, 2011.

[10] F. Folino, M. Guarascio, and L. Pontieri, “Discovering high-level performance models for ticket resolution processes,” in *On the Move to Meaningful Internet Systems: OTM 2013 Conferences*, R. Meersman, H. Panetto, T. Dillon, J. Eder, Z. Bellahsene, N. Ritter, P. De Leenheer, and D. Dou, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 275–282.

[11] M. Polato, A. Sperduti, A. Burattin, and M. de Leoni, “Time and activity sequence prediction of business process instances,” *Computing*, pp. 1–27, 2018.

[12] G. T. Lakshmanan, D. Shamsi, Y. N. Doganata, M. Unuvar, and R. Khalaf, “A markov prediction model for data-driven semi-structured business processes,” *Knowledge and Information Systems*, vol. 42, no. 1, pp. 97–126, Jan 2015.

[13] N. Tax, I. Verenich, M. L. Rosa, and M. Dumas, “Predictive business process monitoring with LSTM neural networks,” in *CAiSE 2017*, ser. LNCS, vol. 10253. Springer, 2017, pp. 477–492.

[14] A. Rogge-Solti, W. M. van der Aalst, and M. Weske, “Discovering stochastic Petri nets with arbitrary delay distributions from event logs,” in *BPM Workshops 2013*, ser. LNBIP, vol. 171. Springer, 2014, pp. 15–27.

[15] A. Senderovich, A. Rogge-Solti, A. Gal, J. Mendling, A. Mandelbaum, S. Kadish, and C. A. Bunnell, “Data-driven performance analysis of scheduled processes,” in *BPM 2015*, ser. LNCS, vol. 9253. Springer, 2015, pp. 35–52.

[16] A. Rogge-Solti and M. Weske, “Prediction of business process durations using non-markovian stochastic petri nets,” *Inf. Syst.*, vol. 54, pp. 1–14, 2015.

[17] A. Senderovich, M. Weidlich, A. Gal, and A. Mandelbaum, “Queue mining for delay prediction in multi-class service processes,” *Inf. Syst.*, vol. 53, pp. 278–295, 2015.

[18] A. Cuzzocrea, F. Folino, M. Guarascio, and L. Pontieri, “Predictive monitoring of temporally-aggregated performance indicators of business processes against low-level streaming events,” *Information Systems*, vol. 81, pp. 236 – 266, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0306437917301023

[19] C. Malandri, M. Briccoli, L. Mantecchini, and F. Paganelli, “A discrete event simulation model for inbound baggage handling,” *Transportation Research Procedia*, vol. 35, pp. 295 – 304, 2018, iNAIR 2018.

[20] T. G. Dietterich, “Machine learning for sequential data: A review,” in *Structural, Syntactic, and Statistical Pattern Recognition*, T. Caelli, A. Amin, R. P. W. Duin, D. de Ridder, and M. Kamel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 15–30.