Proceedings

2020 2nd International Conference on Process Mining

ICPM 2020

Proceedings

2020 2nd International Conference on Process Mining

ICPM 2020

Virtual Conference
4-9 October 2020

Editors
Boudewijn van Dongen
Marco Montali
Moe Thandar Wynn

IEEE COMPUTER SOCIETY CONFERENCE PUBLISHING SERVICES

Los Alamitos, California
Washington • Tokyo

IEEE COMPUTER SOCIETY

ISBN-13: 978-1-7281-9832-3
BMS Part# CFP20S62-ART

Additional copies may be ordered from:

IEEE Computer Society
Customer Service Center
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1314
Tel: + 1 800 272 6657
Fax: + 1 714 821 4641
http://computer.org/cps
cps@computer.org

IEEE Service Center
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
Tel: + 1 732 981 0060
Fax: + 1 732 981 9667
http://shop.ieee.org/store/
customer-service@ieee.org

IEEE Computer Society
Asia/Pacific Office
Watanabe Bldg., 1-4-2
Minami-Aoyama
Minato-ku, Tokyo 107-0062
JAPAN
Tel: + 81 3 3408 3118
Fax: + 81 3 3408 3553
tokyo.ofc@computer.org

Editorial production by Cristina Ceballos
Cover art production by Hector Torres

IEEE Computer Society
***Conference Publishing Services* (CPS)**
http://www.computer.org/cps

2020 2nd International Conference on Process Mining (ICPM)

ICPM 2020

Table of Contents

Online Operational Support

Time and Predictions

Data Quality and Preparation

Discovery with Unconventional Input

Conformance Checking

Rule Mining

Process Discovery

Anomaly Detection and Clustering

Message from the General Chairs
ICPM 2020

Welcome to the second International Conference on Process Mining (ICPM 2020). We are very glad to organize this outstanding event, because ICPM is the flagship conference series of Process Mining, supported by the IEEE Task Force on Process Mining.

The academia started talking about Process Mining around 15-20 years ago, and industry recognized its value a few years later. The momentum currently gained by Process Mining is unsurprising because, compared with traditional Process Intelligence, Process Mining is based on evidence, yielding direct and tangible process performance improvements. Furthermore, Process Mining is widely applicable: processes to monitor and optimize are everywhere, such as those in healthcare, financial and educational institutes, including for providing services to the customers, and many more. In fact, the COVID-19 outbreak and the consequent lockdown have made more evident that improvised and ad-hoc processes are not viable, and highlighted the importance to standardize, automatize, monitor and optimize the processes, namely what Process Mining ultimately aims at.

ICPM 2020 was originally planned to be in Padua (Italy), which hosts one the oldest universities in the world, founded in 1222, where Galileo was a lecturer between 1592 and 1610.

The COVID-19 outbreak forced us to change the plan, and ensure a safe and careful run of the conference. This brought us to the decision to run ICPM 2020 as a virtual conference. The possibility to simply postpone the conference to 2021 has never been taken into a serious consideration, because research, industrialization and, more generally, a sustainable progress should not stop.

ICPM 2020 aims to be a lively event where not only research and practice advances in the field are presented, but also discussions and e-networking are fostered, without requiring travel from the attendees. This allows the participants from anywhere in the world to actively contribute to ICPM, despite the exceptional circumstances of the COVID-19 outbreak.

The major Process Mining leaders have already confirmed their presence at the conference, both from the academia and the industry sector. This is splendid, something which we are very proud of: it will make ICPM 2020 a remarkable venue where these two sectors meet, give feedback and provide a boost to each other. We are delighted that this spirit has been recognized by many businesses and vendors, which have been willing to support and sponsor ICPM 2020. We thank every sponsor for the support, which especially invaluable in the current circumstances.

The program is very rich, running over a period of six days, from Sunday, October 4th to Friday, October 9th. It is also very diversified to embrace academia, industry, and software vendors.

Three days of the program are devoted to the research track, i.e. Tuesday, Thursday and Friday, where 23 academics will present their latest Process Mining research.

On Wednesday, ICPM 2020 will feature an industry day of excellent speakers who will report on their experience on applying Process Mining in their companies. The industry day will be complemented on Thursday by a panel.

Other co-located initiatives are the "cherry on top": ICPM includes (i) several tool demonstrations (Tuesday and Thursday), (ii) three Process Mining contests (Monday), (iii) a doctoral consortium (Sunday), to pave the research roads of new Process Mining researchers, as well as (iv) a series of exciting workshops (Monday) to delineate the future Process Mining research. The workshops have been a novel addition to this conference edition with respect to ICPM 2019, which has certainly been a success, judging the excellent response in terms of number of submitted papers. All of this makes ICPM 2020 "the virtual place to be" for anyone working on Process Mining! Enjoy the conference!

We could not complete this message without thanking the people who made ICPM 2020 possible. We start from the chairs, jury and committee members of the different tracks and initiatives: they all performed an excellent job, and worked in concert with us to prepare an excellent, high-quality program.

We conclude acknowledging the main members of the local team at University of Padua: (i) the secretaries at our Department of Mathematics, who supported us with the conference finance and orders, (ii) Dr. Merylin Monaro, who helped us with many organization points, and (iii) Dr. Sahel Abdinia, who built up and configured the virtual conference system. They provided us with an invaluable support to organize the conference, even during the hard time of the COVID-19 lock-down.

Massimiliano de Leoni
Alessandro Sperduti
University of Padua, Italy
ICPM 2020 General Chairs

Message from the Program Chairs
ICPM 2020

Welcome to the second International Conference on Process Mining (ICPM 2020).

The second edition of the Process Mining Conference received 72 full research paper submissions. Each paper was reviewed by at least three Program Committee members; which was followed by a meta-review discussion among the three program co-chairs. At the end of the process, we accepted 23 high-quality papers (the acceptance rate of 32%). We believe we have compiled a solid research program across three days, split into eight sessions corresponding to major areas of research within the field of process mining, such as automated process discovery, conformance checking, anomaly detection and clustering, operational support, predictive analytics, data-preprocessing and analysis, and applications.

The research track of the conference started on Tuesday with a keynote by Professor Thomas Seidl who is a Professor of Computer Science at Ludwig-Maximilians-Universität München (LMU Munich), Germany. Professor Seidl shared his views on the relationship between the fields of data mining and process mining.

Other activities were carried out in parallel with the scientific program. First, on Sunday, we held the Doctoral Consortium which offered a great opportunity to our PhD students to seek early feedback on their process mining research. Six workshops on a range of process mining topics and applications were hosted on Monday. An Industry Day was held on Wednesday which included a rich showcase of process mining applications by industry leaders. On Thursday, the academic tool demonstrations were held. We also announced the winners of the Process Discovery Contest, the BPI Challenge, the Conformance Checking Contest, and the Best Process Mining Dissertation Award. On Friday, we hosted four final sessions of research papers showcasing the latest findings in process mining research.

We would like to thank all those that helped making this virtual conference a wonderful successful story even during this challenging time of COVID-19. First and foremost, the University of Padua, for hosting the conference online and managing all the logistics involved. Second, the Program Committee members that participated 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 over a period of six days. And last, we would of course like to thank all our sponsors for their support.

Boudewijn van Dongen
Marco Montali
Moe Thandar Wynn
ICPM 2020 Program Co-Chairs

Conference Organization
ICPM 2020

Local Organizing Committee
Sahel Abdinia
Davide Bresolin
Massimiliano de Leoni
Merylin Monaro
Nicolò Navarin
Alessandro Sperduti

IEEE Task Force Steering Committee
Wil van der Aalst, *RWTH Aachen University and Fraunhofer FIT, Germany*
Rafael Accorsi, *PricewaterhouseCoopers, Zürich, Switzerland*
Andrea Burattin, *Technical University of Denmark, Lyngby, Denmark*
Josep Carmona, *Universitat Politècnica de Catalunya, Barcelona, Spain*
Boudewijn van Dongen, *Eindhoven University of Technology, Eindhoven, The Netherlands*
Claudio Di Ciccio, *Sapienza University of Rome, Rome, Italy*
Mieke Jans, *Hasselt University, Hasselt, Belgium*
Julian Lebherz, *Deloitte, Düsseldorf, Germany*
Marco Montali, *Free University of Bozen - Bolzano, Italy*
Marcello La Rosa, *University of Melbourne, Melbourne, Australia*
Jochen de Weerdt, *KU Leuven, Leuven, Belgium*
Moe Thandar Wynn, *Queensland University of Technology, Brisbane, Australia*

Workshop Chairs
Sander Leemans
Henrik Leopold

Tool Demo Chairs
Chiara Di Francescomarino
Jorge Munoz-Gama
Jochen De Weerdt

Industry Chairs
Marc Kerremans
Marcello La Rosa

Doctoral Consortium Chairs
Benoit Depaire
Claudio Di Ciccio

Process Mining Dissertation Award
Marco Montali

Chair International Business Process Intelligence Challenge
Boudewijn van Dongen

Organizers Process Discovery Contest

Josep Carmona
Eric Verbeek
Benoit Depaire

Organizers Conformance Checking Challenge

Abel Armas-Cervantes
Artem Polyvyanyy
Gert Janssenswillen
Luciano García-Bañuelos

Program Committee
ICPM 2020

Robert Andrews, *Queensland University of Technology, Australia*
Abel Armas Cervantes, *The University of Melbourne, Australia*
Ahmed Awad, *University of Tartu, Estonia*
Hyerim Bae, *Pusan National University, South Korea*
Amin Beheshti, *Macquarie University, Australia*
Robin Bergenthum, *FernUniversität in Hagen, Germany*
Andrea Burattin, *Technical University of Denmark, Denmark*
Cristina Cabanillas, *Vienna University of Economics and Business, Austria*
Josep Carmona, *Universitat Politècnica de Catalunya, Spain*
Paolo Ceravolo, *Università degli Studi di Milano, Italy*
Thomas Chatain, *LSV, ENS Paris-Saclay, Cachan, France*
Benjamin Dalmas, *Mines Saint-Etienne, France*
Johannes De Smedt, *The University of Edinburgh, UK*
Jochen De Weerdt, *Katholieke Universiteit Leuven, Belgium*
Pavlos Delias, *Eastern Macedonia and Thrace Institute of Technology, Macedonia*
Benoît Depaire, *Hasselt University, Belgium*
Claudio Di Ciccio, *Sapienza University of Rome, Italy*
Chiara Di Francescomarino, *Fondazione Bruno Kessler-IRST, Italy*
Remco Dijkman, *Eindhoven University of Technology, the Netherlands*
Marlon Dumas, *University of Tartu, Estonia*
Joerg Evermann, *Memorial University of Newfoundland, Canada*
Dirk Fahland, *Eindhoven University of Technology, the Netherlands*
Carlos Fernandez-Llatas, *Universitat Politècnica de València, Spain*
Francesco Folino, *ICAR-CNR, Italy*
Walid Gaaloul, *Computer Science Department Télécom SudParis, France*
Luciano García-Bañuelos, *Tecnológico de Monterrey, Mexico*
Gianluigi Greco, *University of Calabria, Italy*
Daniela Grigori, *Laboratoire LAMSADE, University Paris-Dauphine, France*
Antonella Guzzo, *University of Calabria, Italy*
Mieke Jans, *Hasselt University, Belgium*
Gert Janssenswillen, *Hasselt University, Belgium*
Anna Kalenkova, *The University of Melbourne, Australia*
Christopher Klinkmüller, *Data61|CSIRO, Australia*
Agnes Koschmider, *Keil University, Germany*
Marcello La Rosa, *The University of Melbourne, Australia*
Manuel Lama Penin, *University of Santiago de Compostela, Spain*
Sander J.J. Leemans, *Queensland University of Technology, Australia*
Henrik Leopold, *Kühne Logistics University, Germany*
Irina Lomazova, *National Research University Higher School of Economics, Russia*
Xixi Lu, *Utrecht University, the Netherlands*
Fabrizio Maria Maggi, *Free University of Bozen-Bolzano, Italy*
Felix Mannhardt, *Norwegian University of Science and Technology, Norway*
Niels Martin, *Hasselt University, Belgium*
Raimundas Matulevicius, *University of Tartu, Estonia*
Massimo Mecella, *Sapienza University of Rome, Italy*
Jan Mendling, *Wirtschaftsuniversität Wien, Austria*

Hamid Motahari, *EY AI Lab Palo Alto, USA*
Jorge Munoz-Gama, *Pontificia Universidad Católica de Chile, Chile*
Nicolo' Navarin, *University of Padua, Italy*
Artem Polyvyanyy, *The University of Melbourne, Australia*
Luigi Pontieri, *National Research Council of Italy – CNR, Italy*
Hajo A. Reijers, *Utrecht University, the Netherlands*
Manuel Resinas, *University of Seville, Spain*
Arik Senderovich, *University of Toronto, Canada*
Marcos Sepúlveda, *Pontificia Universidad Católica de Chile, Chile*
Pnina Sofer, *University of Haifa, Israel*
Minseok Song, *Pohang University of Science and Technology, South Korea*
Arthur ter Hofstede, *Queensland University of Technology, Australia*
Wil van der Aalst, *RWTH Aachen University, Germany*
Boudewijn van Dongen, *Eindhoven University of Technology, the Netherlands*
Sebastiaan J. van Zelst, *Fraunhofer Institute – FIT & RWTH Aachen University, Germany*
Seppe Vanden Broucke, *Katholieke Universiteit Leuven, Belgium*
Eric Verbeek, *Eindhoven University of Technology, the Netherlands*
Jianmin Wang, *Tsinghua University, China*
Barbara Weber, *University of St. Gallen, Switzerland*
Ingo Weber, *TU Berlin, Germany*
Matthias Weidlich, *Humboldt-Universität zu Berlin, Germany*
Moe Wynn, *Queensland University of Technology, Australia*

Additional Reviewers
ICPM 2020

Mehdi Acheli
Nour Assy
Massimo Guarascio
Richard Hobeck
Dominik Janssen
Alexey A. Mitsyuk
Luise Pufahl
Lluís Padró
Farbod Taymouri

Sponsors
ICPM 2020

Platinum

Silver

Contributing Organization

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Keynote

Data Mining on Process Data

Thomas Seidl

Abstract:

Data Mining and Process Mining -- is one just a variant of the other, or do worlds separate the two areas from each other? The notions sound so similar but the contents sometimes look differently, so respective researchers may get confused in their mutual perception, be it authors or reviewers. The talk recalls commonalities like model-based supervised and unsupervised learning approaches, and it also sheds light to peculiarities in process data and process mining tasks as seen from a data mining perspective. When considering trace data from event log files as time series, as sequences, or as activity sets, quite different data mining techniques apply and may be extended and improved.

A particular example is rare pattern mining, which fills a gap between frequent patterns and outlier detection. The task aims at identifying patterns that occur with low frequency but above single outliers. Structural deficiences may cause malfunctions or other undesired behavior which get discarded as outliers in event logs, since they are observed infrequently only. Rare pattern mining may identify these situations, and recent approaches include clustering or ordering non-conformant traces.The talk concludes with some remarks on how to sell process mining papers to the data mining community, and vice versa, in order to improve mutual acceptance, and to increase synergies in the fields.

Bio:

Thomas Seidl is professor for computer science and head of the Database Systems and Data Mining group at LMU Munich. He is co-chair of the nationally funded Munich Center for Machine Learning (MCML), of LMU's data science lab and of LMU's master program in data science. Seidl also serves as member of Leibniz Supercomputing Center's directorate, of many program committees and other scientific boards. His fundamental research on data mining and database technologies with applications in engineering, business, life sciences and humanities yielded more than 300 scientific publications so far. Before returning to Munich in 2016, Seidl held a chair in Computer Science at RWTH Aachen University from 2002, after he had finished his Master, PhD and habilitation in CS in Munich at TUM and LMU, respectively.

Explainable Predictive Process Monitoring

Riccardo Galanti*‡, Bernat Coma-Puig†, Massimiliano de Leoni‡, Josep Carmona†, and Nicolò Navarin‡
*myInvenio, Reggio Emilia, Italy, ‡University of Padua, Padua, Italy, †Universitat Politècnica de Catalunya, Barcelona, Spain
Email: riccardo.galanti@my-invenio.com, {deleoni,nnavarin}@math.unipd.it
{bcoma,jcarmona}@cs.upc.edu

***Abstract*—Predictive Business Process Monitoring is becoming an essential aid for organizations, providing online operational support of their processes. This paper tackles the fundamental problem of equipping predictive business process monitoring with explanation capabilities, so that not only the *what* but also the *why* is reported when predicting generic KPIs like remaining time, or activity execution. We use the game theory of Shapley Values to obtain robust explanations of the predictions. The approach has been implemented and tested on real-life benchmarks, showing for the first time how explanations can be given in the field of predictive business process monitoring.**

I. Introduction

Within the field of Process Mining, predictive monitoring aims to forecast the running process instances with the purpose of timely signalling those that require special attention (those that may take too long, cost too much, not be satisfactory, etc.). Several approaches have been proposed in literature to deal with predictive monitoring (cf. Section III-A and the survey by Márquez et al. [9]), which has received significant attention in the last years. However, the majority of these approaches rely on black-box models (e.g. based on LSTM, i.e. Long Short-Term Memory neural models), which are proven to be more accurate, at the cost of being unable to provide a feedback to the user. On the other hand, approaches based on explicit rules (e.g. based on classification/regression trees) tend to be significantly less accurate. While the priority remains on giving accurate predictions, users need to be provided with an explanation of the reason why a given process execution is predicted to behave in a certain way. Otherwise, users would not trust the model, and hence they would not adopt the predictive-monitoring technology [13], [4].

This paper tackles the problem of equipping process monitoring with explanations of the predictions. It leverages on current state of the art of Explainable AI (cf. Section III-B), defining a framework for explainable process monitoring of generic KPIs. The proposed framework is independent of the machine- or deep-learning technique that is employed to make the predictions. However, we aim to instantiate the framework to prove its effectiveness. With this aim in mind, we built a process-monitoring framework, based on LSTM models, that is also able to explain any generic KPI, numerical or nominal. In a nutshell, given a running case, our framework estimates the future KPI value and returns the set of attributes that influence its prediction the most.

Experiments were conducted on different benchmarks, including the real-life process of an Italian financial institute, with the aim of predicting different KPIs, namely remaining time, costs, and the eventual occurrence of certain undesired activities. Explanations can be generated at LSTM-model level, to be provided to process stakeholders to understand the general trend of the model, but also at run-time, to explain the predictions of each single running case. The explanations obtained for the aforementioned financial institute are in line with those of the analysts of the process. The remarkable difference is that our results were obtained within a few days of automatic computations, instead of long analyses.

The rest of the paper is organized as follows. Section II states the problem addressed in this paper. Section III summarizes the most relevant work related to process predictive monitoring and Explainable AI. Section IV sketches the state of the art on using LSTM models for predictive monitoring, on which we build to provide explanations. Section V reports on our framework for explainable predictive process monitoring. Section VI reports on our framework's operationalization, and on the case studies conducted with an Italian financial institute, whereas Section VII concludes the paper.

II. Problem Statement

The starting point for a prediction system is an *event log*. An event log is a multiset of *traces*. Each trace describes the life-cycle of a particular *process instance* (i.e., a *case*) in terms of the *activities* executed and the process *attributes* that are manipulated.

Definition 2.1 (Events): Let $\mathcal{A}$ be the set of process attributes. Let $\mathcal{W}_{\mathcal{A}}$ be a function that assigns a domain $\mathcal{W}_{\mathcal{A}}(a)$ to each process attribute $a \in \mathcal{A}$. Let be $\overline{\mathcal{W}} = \cup_{a \in \mathcal{A}} \mathcal{W}_{\mathcal{A}}(a)$. An *event* $e \in \mathcal{E}$ is a partial function $\mathcal{A} \not\rightarrow \overline{\mathcal{W}}$ assigning values to process attributes, with $e(a) \in \mathcal{W}_{\mathcal{A}}(a)$.

Note that the same event can potentially occur in different traces, namely attributes are given the same assignment in different traces. This means that potentially the entire same trace can appear multiple times. This motivates why an event log is to be defined as a multiset of traces.[1]

Definition 2.2 (Traces & Event Logs): Let $\mathcal{E}$ be the universe of events. A trace σ is a sequence of events, i.e. $\sigma \in \mathcal{E}^*$. An event-log L is a multiset of traces, i.e. $L \subset \mathbb{B}(\mathcal{E}^*)$.

Predictive monitoring aims to estimate the future KPI values of the running cases. Here, we aim to be generic, meaning that KPIs can be of any nature:

[1]Given a set X, $\mathbb{B}(X)$ indicates the set of all multisets with the elements in X.

978-1-7281-9832-3/20/$31.00 ©2020 IEEE
DOI 10.1109/ICPM49681.2020.00012

Definition 2.3 (KPI): Let $\mathcal{E}$ be the universe of events defined over a set $\mathcal{A}$ of attributes. Let $\mathcal{W}_K$ be the domain of the KPI values. A KPI is a function $\mathcal{T} : \mathcal{E}^* \times \mathbb{N} \not\to \mathcal{W}_K$ such that, given a trace $\sigma \in \mathcal{E}^*$ and an integer index $i \leq |\sigma|$, $\mathcal{T}(\sigma, i)$ returns the KPI value of σ after the occurrence of the first i events.[2]

Note that our KPI definition assumes it to be computed a posteriori, when the execution is completed and leaves a complete trail as a certain trace σ. In many cases, the KPI value is updated after each activity execution, which is recorded as next event in trace; however, other times, this is only known after the completion. We aim to be generic and account for all relevant cases. Given a trace $\sigma = \langle e_1, \ldots, e_n \rangle$ that records a complete process execution, the following are three potential KPI definitions:

Remaining Time. $\mathcal{T}_{remaining}(\sigma, i)$ is equal to the difference between the timestamp of e_n and that of e_i.

Activity Occurrence. It measures whether a certain activity is going to eventually occur in the future, such as an activity *Open Loan* in a loan-application process. The corresponding KPI definition for the occurrence of an activity A is $\mathcal{T}_{occur\_A}(\sigma, i)$, which is equal to true if activity A occurs in $\langle e_{i+1}, \ldots, e_n \rangle$ and $i < n$; otherwise false.

Customer Satisfaction. This is a typical KPI for several service providers. Let us assume, without losing generality, to have a trace $\sigma = \langle e_1, \ldots, e_n \rangle$ where the satisfaction is known at the end, e.g. through a questionnaire. Assuming the satisfaction level is recorded with the last event - say $e_n(sat)$. Then, $\mathcal{T}_{cust\_satisf}(\sigma, i) = e_n(sat)$.

The following definition states the prediction problem:

Definition 2.4 (The Prediction Problem): Let L be an event log that records the execution of a given process, for which a KPI $\mathcal{T}$ is defined. Let $\sigma = \langle e_1, \ldots, e_k \rangle$ be the trace of a running case, which eventually will complete as $\sigma_T = \langle e_1, \ldots, e_k, e_{k+1} \ldots, e_n \rangle$. The prediction problem can be formulated as forecasting the value of $\mathcal{T}(\sigma_T, i)$ for all $k < i \leq n$.

As indicated in Section I, we aim to provide an explanation for the predictions. In particular, for each running case, we aim to return the set of attributes influencing its prediction the most, with the corresponding magnitude and the indication whether the attributes increase or decrease the predicted KPI's value.

In the light of the above, for each trace $\sigma = \langle e_1, \ldots, e_n \rangle$, the problem can be stated as finding a function $\mathcal{K}_{(\sigma,\mathcal{T})}$ such that, for all $a \in \mathcal{A}$, $v \in \mathcal{W}_{\mathcal{A}}(a)$, and for all i s.t. $-n < i \leq 0$, $\mathcal{K}_{(\sigma,\mathcal{T})}(a, v, i)$ is different from zero if and only if the assignment of value v to attribute a by $e_{(n-i)}$ has influenced the prediction of KPI $\mathcal{T}$. The absolute value of $\mathcal{K}_{(\sigma,\mathcal{T})}(a, v, i)$ indicates how much this influence is, where a zero value indicates no influence. If $\mathcal{K}_{(\sigma,\mathcal{T})}(a, v, i) \neq 0$, its positive or negative sign indicates whether the influence is towards increasing or decreasing the KPI value:

[2]Given a sequence X, $|X|$ indicates the length of X. Notation $\not\to$ indicates that the function is partial.

Definition 2.5 (The Prediction-Explanation Problem): Let L be an event log over a set $\mathcal{A}$ of attributes, with domains $\mathcal{W}_{\mathcal{A}}$. Let $\sigma = \langle e_1, \ldots, e_k \rangle$ be a running case with a KPI definition $\mathcal{T}$. Let be $\overline{\mathcal{W}} = \cup_{a \in \mathcal{A}} \mathcal{W}_{\mathcal{A}}(a)$. Explaining the prediction is the problem of computing a function $\mathcal{K}_{(\sigma,\mathcal{T})} : \mathcal{A} \times \overline{\mathcal{W}} \times [-k+1, 0] \to \mathbb{R}$, where $v \notin \mathcal{W}_{\mathcal{A}}(a) \Rightarrow \mathcal{K}_{(\sigma,\mathcal{T})}(a, v, i) = 0$.

III. Related Works

A. Prediction of Process-Related KPIs

The predictive-monitoring survey of Márquez et al. [9] reports on the large repertoire of techniques and tools that were developed to address this problem. However, the authors claim that *"little attention has been given to [...] explaining the prediction values to the users so that they can determine the best way to act upon"*, and that *"it is necessary to develop tools that help users to query these models in order to get information that is relevant for them"*. These are in fact the problems tackled in this paper, so as to ensure that the predictive-monitoring system is trusted, and thus used.

Predictive monitoring has been built on different machine and deep-learning techniques, and also on their ensemble [9]. Different research works have recently illustrated that the so-called Long Short-Term Memory networks (LSTMs) generally outperform other methods (see, e.g., [14], [23], [12]). Therefore, while our explanation framework is independent of the machine- or deep-learning technique that is employed, we operationalize it with LSTMs. Section IV provides further details on LSTMs, and details how they are employed for business-process predictive monitoring.

It was explained above that little research work has been conducted on explaining the outcome of process predictive monitoring. The most relevant work is by Rehse et al. [15], which also aims at providing a dashboard to process participants with predictions and their explanation. However, the paper does not provide sufficient details on the actual usage of the explainable-AI literature, and the very preliminary evaluation is based on one single artificial process that consists of a sequence of five activities. Breuker et al. also try to tackle the problem [3], but their attempt is not independent of the actual technique employed for predictions. Furthermore, their explanations are only based on the activity names, while the explanations can generally involve resources, time, and more (cf. the case studies reported in Section VI).

B. Explanation of Machine-Learning Models

Few approaches exist in the literature to explain machine learning models, arisen from the need to understand complex black-box algorithms like ensembles of Decision Trees and Deep Learning [16], [20], [22], [7].

The adoption of explanatory methods in industry is at an early stage; in [21] an approach of fake news detection grounded in explainability is introduced. A significant amount of work in literature is focused on healthcare applications. We highlight [8], an implementation of the Shapley Values in healthcare, where the explanatory method is used to prevent

hypoxaemia during surgery, and [10], where explainability is used for analysis of patience re-admittance.

The SHAP implementation of the Shapley values for Deep Learning has the strong theoretical foundation of the original game theory approach, with the advantage of providing offline explanations that are consistent with the online explanations. Moreover, SHAP avoids the problems in consistency seen in other explanatory approaches (e.g. the lack of robustness seen in the online surrogate models, as analysed in [1]). The framework proposed in this paper specializes the use of Shapley values to the problem of providing explanations for predictive analytics.

We also considered attention mechanisms [2] as an alternative. However, two limitations made us opt for Shapley values. First, attention mechanisms necessarily have to be integrated in a Neural Network architecture, while Shapley values can be applied to any Machine or Deep Learning algorithm. The second limitation is linked to the lack of consensus that attention weights are always correlated to feature importance. Jain et al. [17] find it *"at best, questionable – especially when a complex encoder is used, which may entangle inputs in the hidden space"*, Serrano et al. [18] state that *"attention weights often fail to identify the sets of representations most important to the model's final decision"*.

IV. The Use of LSTMs for Predictive Monitoring

As indicated in Section I, we implemented our framework by leveraging on LSTM models [6], a special type of Recurrent Neural Networks. LSTM models natively support the predictions where the independent variables are sequences of elements, and the literature has shown that they are among the most suitable methods for predictive business monitoring (cf. Section III-A).

The construction of LSTM models fall into the problem of supervised learning, which aims to learn the model from a training set, for which the value of the dependent variable is known. This set is composed by pairs $(x, y) \in \mathcal{X} \times \mathcal{Y}$ where $\mathcal{X}$ represents the independent variables with their values (also known as **features**), and $\mathcal{Y}$ is the value observed for the dependent variable (i.e. the value we aim to predict).

In the domain of LSTM learning, $\mathcal{X}$ consists of sequences of vectors with a certain number n of dimensions, i.e. $\mathcal{X} = (\mathbb{R}^n)^*$.[3] When LSTM is used for predictive business monitoring using KPI values in a domain $\mathcal{W}_K$ (cf. Definition 2.3), $\mathcal{Y}$ is $\mathcal{W}_K$.

With these preliminaries at hand, we built a process monitoring framework composed by two phases: off-line and on-line.

The off-line phase requires an event log L and a KPI definition $\overline{\mathcal{T}}$ as input. This enables creating the dataset for training and testing the LSTM model, which consists of pairs $(x, y) \in (\mathbb{R}^n)^* \times \mathcal{W}_K$. The input is, hence, a sequence of vectors; conversely, a trace is a sequence of events. Therefore, each event needs to be encoded as a vector, which is a problem largely studied: we use the same encoding as in [12]; this can be abstracted as an **event-to-vector encoding function** $\rho : \mathcal{E} \to \mathbb{R}^n$. In a nutshell, each numeric attribute a of event e becomes a different dimension of $\rho(e)$, which takes on value $e(a)$. Each boolean attribute a is also a different dimension, with either 0 or 1 depending whether $e(a)$ is false or true. Each literal attributes a is represented through the so-called *one-hot encoding*: one different dimension exists for each value $v \in \mathcal{W}_{\mathcal{A}}(a)$, and the dimension referring to value $e(a)$ takes on value 1, with the other dimensions be assigned value 0. Function ρ can also be overloaded to traces: $\rho(\langle e_1, \ldots, e_m \rangle) = \langle \rho(e_1), \ldots, \rho(e_m) \rangle$.

The dataset is created starting from each prefix σ' of each trace $\sigma \in L$: σ' will generate one item in the data set consisting of a pair $(x, y) \in (\mathbb{R}^n)^* \times \mathcal{W}_K$ where $x = \rho(\sigma')$ and $y = \overline{\mathcal{T}}(\sigma, |\sigma'|)$. The dataset is later divided in one larger part for training the LSTM model, and a smaller part for testing. The test part is used to evaluate the quality of the LSTM model, in terms of different metrics. Details of the proportions and the quality metrics employed are discussed in Section VI. The **LSTM-based process predictor** trained from a dataset $\mathcal{D} \subset (\mathbb{R}^n)^* \times \mathcal{W}_K$ can be abstracted as a function $\Phi_{\mathcal{D}} : \mathbb{R}^{n^*} \to \mathcal{W}_K$.

The on-line phase aim is to predict the KPI of interest for a set of running cases of the process, identified by a set L' of partial traces (i.e., a log). It relies on the LSTM-based process predictor $\Phi_{\mathcal{D}}$: for each $\sigma' \in L'$, the predicted KPI value is $\Phi_{\mathcal{D}}(\rho(\sigma'))$.

V. Explanation of Generic KPI Predictions

This section reports on the main contribution of this paper, namely using Shapley Values to explain the predictions of any predictive model.

Section V-A introduces the theory behind Shapley values, while Section V-B illustrates its application and adaptation for predictive process monitoring. Then, in Section V-C we provide the general picture and the two main types of explanations reported.

A. The Theory of Shapley Values

The Shapley Values [19] is a game theory approach to fairly distribute the payout among the players that have collaborated in a cooperative game. This theory can be adapted as an approach to explain a predictive model. The assumption is that the features from an instance correspond to the players, and the payout is the difference between the prediction made by the predictive model and the average prediction (later referred to as the *base value*). Intuitively, given a predicted instance, the Shapley Value of a feature expresses how much the feature value contributes to the model prediction [11]:

Definition 5.1 (Shapley Value): Let $X = \{x_1, \ldots, x_n\}$ be a set of features. The Shapley value for feature x_i is defined as:

$$\psi_i = \sum_{S \subseteq \{x_1, \ldots, x_m\} \setminus \{x_i\}} \frac{|S|!(p-|S|-1)!}{p!} \left(val\left(S \cup \{x_i\}\right) - val(S) \right)$$

where $val(T)$ is the so-called payout for only using the set of feature values in $T \subset X$ in making the prediction.

[3] In literature, LSTMs are often trained on the basis of matrices. However, a sequence of m vectors in $\mathbb{R}^n$ can be seen, in fact, as a matrix in $\mathbb{R}^{n,m}$. We use here the dataset representation as vectors to simplify the formalization.

Intuitively, the formula in Definition 5.1 evaluates the effect of incorporating the feature value x_i into any possible subset of the feature values considered for prediction. In the equation, variable S runs over all possible subsets of feature values, the term $val\,(S \cup \{x_i\}) - val(S)$ corresponds to the marginal value of adding x_i in the prediction using only the set of feature values in S, and the term $\frac{|S|!(p-|S|-1)!}{p!}$ corresponds to all the possible permutations with subset size $|S|$, to weight different sets differently in the formula. This way, all possible subsets of attributes are considered, and the corresponding effect is used to compute the Shapley Value of x_i.

B. Explainable Predictions through Shapley Values

The starting point is a event-to-vector encoding function $\rho : \mathcal{E} \rightarrow \mathbb{R}^n$ that maps each event to a feature vector (cf. Section IV). Given an event e_i, $\rho(e_i) = [x_i^1, \ldots, x_i^n]$ where each feature x_i^j is associated with an event attribute a_i^j and, possibly, with a value v_i^j. We mentioned that, if an attribute a_i^j is categorical, we need to introduce as many features as its possible values (one-hot encoding). Namely, x_i^j is both associated with an attribute a_i^j, and with a value v_i^j. If the feature associated with attribute a_i^j and value v_i^j takes on value 1, then $e(a_i^j) = v_i^j$; otherwise, the value is 0. If an attribute a_i^j is conversely numerical, only one feature x_i^j exists with value $e(a_i^j)$. When applied for explainable predictive monitoring, the Shapley values of a trace $\sigma = \langle e_1, \ldots, e_m \rangle$ are computed over the features of the vector $\chi = [x_1^1, \ldots, x_1^n, \ldots, x_m^1, \ldots, x_m^n]$ where $\rho(e_i) = [x_i^1, \ldots, x_i^n]$ for $1 \leq i \leq m$.

When applying Definition 5.1 to all features of χ, the result is a vector of Shapley values $\Psi = [\psi_1^1, \ldots, \psi_1^n, \ldots, \psi_m^1, \ldots, \psi_m^n]$ associated to feature vector χ, and attributes $[a_1^1, \ldots, a_1^n, \ldots, a_m^1, \ldots, a_m^n]$. Any Shapley value ψ_i^j can be either positive or negative. A positive or negative value indicates that the feature contributes to increasing or decreasing the value, respectively.

This allows us to construct the explanations. The first step is to determine which features are relevant and at which timestep.[4] For this, we consider the average μ of the values in Ψ along with their standard deviation ξ. This allows to define an interval $I = [\mu - \delta\xi, \mu + \delta\xi]$ of Shapley values that are not considered to contribute significantly, where δ is a parameter set by the user. This reduces the number of features that are considered in the explanation, limiting its verbosity.

Let us consider each Shapley value $\psi_i^j \notin I$, associated with feature x_i^j and an event's attribute a_i^j.

If a_i^j is a numerical attribute, attribute a_i^j is the explanation itself, i.e. $\forall \overline{v} \in \mathcal{W}_{\mathcal{A}}(a)$. $\mathcal{K}_{(\sigma,\mathcal{T})}(a_i^j, \overline{v}, i-m) = \psi_i^j$.

If a_i^j is a categorical attribute, x_i^j is a one-hot encoded feature, and it is also associated with a value v_i^j. If $x_i^j = 1$, the explanation obtained is that $a_i^j = v_i^j$ contributes to the KPI value: $\mathcal{K}_{(\sigma,\mathcal{T})}(a_i^j, v_i^j, i-m) = \psi_i^j$. Otherwise, $x_i^j = 0$, and the explanation is $a_i^j \neq v_i^j$, namely $\forall \overline{v} \in \mathcal{W}_{\mathcal{A}}(a) \setminus \{v_i^j\}$. $\mathcal{K}_{(\sigma,\mathcal{T})}(a_i^j, \overline{v}, i-m) = \psi_i^j$.

[4]In this context each timestep refers to a different event of the trace along with its attributes (features). For instance, timestep zero refers to the first event of the trace, timestep one to the second, etc.

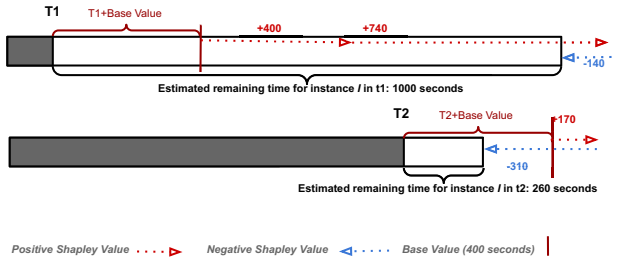


Fig. 1: Two explanations examples using Shapley Values. When the Remaining Time predicted is high (i.e. higher than Base Value), the Shapley Values indicate which features increase the prediction. Similarly, when the prediction is smaller than the Base Value, most of the Shapley Values are negative.

Any other combination (a, v, i) that does not fall into the situations above is such that $\mathcal{K}_{(\sigma,\mathcal{T})}(a, v, i) = 0$.

While an exact computation of the Shapley values requires to consider all combinations of features (hence, the algorithm is exponential on the number of features), efficient estimations can be obtained through polynomial algorithms that use greedy approaches [11].

To conclude, let us illustrate how Shapley values help explain a typical KPI in predictive process monitoring: estimated remaining time. Figure 1 shows the estimated remaining time of the same case in two different moments: T1, when the case started (upper figure, with an estimated remaining time of 1000 seconds), and T2, when it is close to its end (lower figure, with an estimated remaining time of 260 seconds). Considering that the Base Value is 400 seconds, the explanatory method would indicate, at T1, which features have been useful for the predictive model to predict a high value, i.e. the features with a positive Shapley Value. On the other hand, for T2, most of the Shapley Values would be negative, since the model has predicted a value smaller than the base value.

C. Overall Approach for Explaining Generic KPI Predictions

Explanations can be used offline to explain the features/factors that the trained model uses to make predictions, moreover they can be employed online on each running case to put forward the factors that affected the predictions. In particular, offline explanations are calculated on the test dataset, which is a part of the dataset not used for training the model (information about the division between train and test sets will be provided in Section VI).

*1) **Offline Explanations**:* Our offline explanation strategy is to provide an heatmap that overviews the importance of each factor in explaining the instances of the test dataset.

In particular, given an event log L, we consider each prefix σ' of each trace in L. Then, we compute the explanations as defined in Section V-B. Figure 2 shows an example of a heatmap reporting the frequency in which an explanation is relevant after each event in every trace. The y axis lists different explanations of types $attr = value$ or $attr \neq value$

TABLE I: Online explanations for *Remaining Time* for three running cases. When the explanation is followed by (-1), it means that it refers to the value assigned to the attribute by the event that precedes the last of respective case.

CASE ID	REMAINING TIME	Explanations for increasing remaining_time	Explanations for decreasing remaining_time
201810011258	5d 6h 7m	ACTIVITY=Evaluating Request (NO registered letter)	CLOSURE_TYPE!=Inheritance
201810000206	5d 2h 12m	ROLE=DIRECTOR	CLOSURE_TYPE=Bank Recess
201811010829	2d 2h 31m	ROLE!=BACK-OFFICE (-1) AND ACTIVITY!=Service closure Request with BO responsibility (-1)	-
...	...	...	...

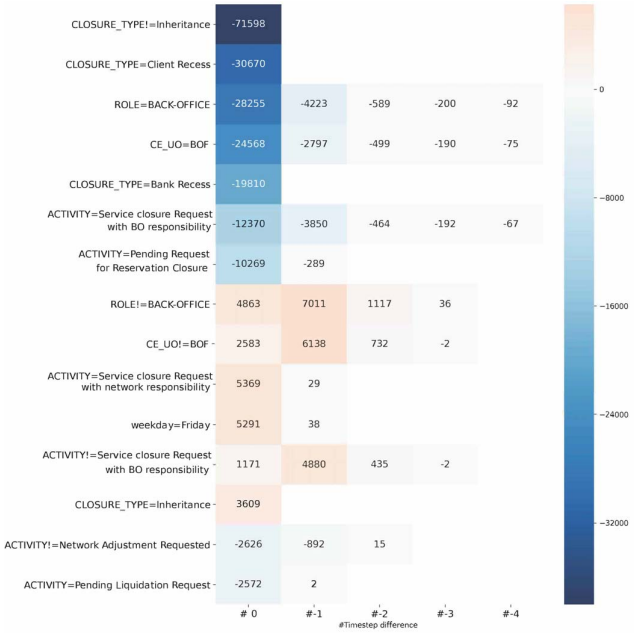


Fig. 2: The offline explanation of the remaining time

while the x axis lists the timestep difference between the event in question and the last event of the considered prefix, namely 0 indicates the last event, -1 indicates the second last, etc. A cell with explanation $a = v$ (y axis) and timestep difference t (x axis) takes on a value $(x - y)$ if there are x prefixes σ' of traces in L s.t. $\mathcal{K}_{(\sigma',\mathcal{T})}(a, v, t) > 0$ and y prefixes σ'' of traces in L s.t. $\mathcal{K}_{(\sigma'',\mathcal{T})}(a, v, t) < 0$. For instance, let us consider the explanation `ROLE=BACK-OFFICE` with timestep difference 0, which is associated with value -28255. This means that -28255 is the difference between the number of prefixes σ' in which `ROLE=BACK-OFFICE` in the last event of σ' contributes to increasing the KPI value and the number of prefixes σ' in which `ROLE=BACK-OFFICE` of last event contributes to decreasing. Similarly, -4223 is the difference when considering the second last event of the prefixes in place of the last. A similar reasoning can be repeated for explanations of type $a \neq v$. The heatmap uses different shades of blue and red to highlight the magnitude of negative and positive values, respectively.

*2) **Online Explanations**:* When we focus on running cases, we generate a table with one row per running case (see, e.g., Table I. Each row shows the case id, unique for each running case, the prediction for the current KPI, and the explanations that influence the prediction. Section VI discusses the case study in detail, including the results in Table I.

VI. Implementation and Experiments

The framework for explainable predictive monitoring has been implemented in Python, using Pandas to elaborate the data, and the shap library[5] to explain the prediction.[6] We relied on Keras framework for the LSTM implementation. The architecture was composed by 8 layers with 100 neurons each.

Each LSTM model was trained in 12-24 hours, and the computation of the off-line explanations (i.e. the heatmaps) required a similar amount of time. For each running case, on-line predictions and explanations are given in ca. half second. Note that training models in less than one day does not pose significant limitations: this is just performed once before putting the system in production.

The remainder of the paper will report on the experiments with different KPIs for the process carried on in an Italian bank. However, we also conducted several additional experiments with publicly-available event logs, which confirm the findings reported here. Space limitation prevent us from reporting on them, which are however discussed in the appendix of the extended version of this paper [5].

A. Domain description

Our assessment is based on the so-called ***Bank Account Closure***, a process executed at an Italian Banking Institution. The process deals with the closure of customer's accounts, which may be requested either by the customer or by the bank, for several reasons.

From the bank's information system, we extracted an event log with 32.429 completed traces and 212.721 events. It contains 15 different activities, 654 possible resources (recorded in an attribute labeled *Ce_Uo*), divided in 3 roles (attribute *role*). Each trace is associated with an attribute *Closure_Type*, which encodes the type of procedure that is carried out for the specific account holder, and the *Closure_Reason*, namely the reason triggering the closure's request. The latter is only known for 79.43% of cases.

For the bank, it is of interest to obtain an estimate of the remaining time until the end for running cases. This allows the bank to decide which cases require special attention, in order to not postpone them too much further. Also, the bank wants

[5]https://shap.readthedocs.io/en/latest

[6]Code can be found at https://github.com/PyRicky/LSTM_Generic_explainable

to be informed whether there are high chances that one or more of the following activities will occur: *Authorization Requested*, *Pending Request for Acquittance of heirs*, and *Back-Office Adjustment Requested*. They are linked to contingency actions, which should be avoided because they would cause inefficiencies in terms of time, costs, and resource utilization. Finally, the bank is also interested in obtaining an estimate of the total cost of a running case, in order to detect in advance which cases require particular attention.

We used two/thirds of the traces as training, and one third as test set. For improving the quality of the trained model, we used hyperparameter optimization, with 20% of the training data employed for this (validation set).

Sections VI-B, VI-C and VI-D report on the outcome for remaining-time prediction, for the prediction of the occurrence of one of those three contingency actions and for total cost prediction, respectively.

B. Results on Remaining Time

Section V showed that the explanation for a learnt prediction model is given as a heatmap during the offline phase. Figure 2 refers to the application for the remaining time prediction. The fact that the closure type is not Inheritance (*Closure_Type!=Inheritance*) is the largest value in the heatmap (as absolute value), so it is the largest factor that influences the prediction. The information that the value is negative (i.e. -71598) indicates that the influence is towards reducing the value, namely towards having lower remaining time. From a domain viewpoint, when the type of procedure is Inheritance, the bank-account holder is passed away. A further analysis of the data confirms this finding: if the type is *Inheritance*, the process duration is 29 days, versus 14 days when the type is different. The evidence in the explanation illustrates that LSTM allowed learning a prediction model that leverages on the closure type to estimate the remaining time. Other important attributes are related to the role associated to the resource and the resource performing each activity. Let us consider attributes *Role=Back-office* and *CE_UO=BOF* that are related to back-office activities, which are generally performed in the final part of cases; it can be seen in the heatmap that even in this case the model is able to predict that the process instance is about to complete (a negative value again indicates smaller remaining time).

The discussion was so far focused on the attribute of the last event. However, the values of attributes of previous events also influence the prediction of remaining time as shown in the heatmaps (see columns related to timestep differences -1, -2, -3 and -4). Consider, e.g., the row *ROLE=Back-office* and column -1: the value -4223 indicates that if the previous event refers to an activity performed by a resource with role Back-office, this influences to lower the prediction: the case is getting even closer to the end. When activities are performed by a resource director the behaviour is considered as exceptional, while activities performed by resources playing the role of applicant are in general performed in the initial part of cases; consequently, the cases usually take longer to complete. This is indicated by the positive value 4863 of the last event in the row *ROLE!=Back-office*, which indicates that the influence is towards increasing the remaining time. Notice that the column related to timestep difference -1 has a bigger value (7011), indicating that if the previous event refers to an exceptional activity, the influence on the prediction will be even stronger. Finally when the activity performed is other than Network Adjustment Requested then the predicted remaining time is smaller; this is in fact an exceptional activity, that only occurs when an error is made in the early stages of the process, and even in this case our framework was able to learn to predict a smaller remaining time when no adjustments need to be done.

Section V indicated that explanations are also given for running cases to explain predictions to process stakeholders. Our implementation returns a CSV file with the predictions for the running cases; a subset is provided in Table I, which shows the factors that increase or decrease the prediction for the remaining time prediction. Let us consider as an example the last row: the remaining time is predicted as being ca. 2 days and 2 hours, with two explanations increasing the prediction, one related to the fact that the previous activity performed was not *Service Closure Request with BO Responsibility*, and the other related to the resource performing the previous activity with a role not being *Back-Office*.

To conclude, since this KPI is numerical and the values are reasonably well balanced, we adopted the *Mean Absolute Error* (MAE), which is the average difference between the actual and the predicted value, computed over all test-set samples. Here, we achieved a MAE of 4.37 days, which is around the 28% of the average case duration (i.e. 15.5 days).

C. Results on Prediction of Activity Occurrence

We mentioned that the financial institute aims to avoid activities related to inefficiencies (e.g. rework): *Pending Request for Acquittance of Heirs*, *Back-Office Adjust Requested* and *Autorization Required*. Space limitation prevents us from showing here all of three: here we focus on activity *Back-Office Adjust Requested*, while the other two are in the appendix complementing the paper [5]. The learnt LSTM model was characterized by an F1 score of 0.65, an Area Under the Receiver Operating Charateristics (AUROC) of 0.86, and an Area under Precision/Recall curve (APR) of 0.69. We computed AUROC and APR, because these metrics are, in fact, more suitable when some classes are unbalanced. This is actually the case for our case study: the three activities are contingency actions, which occur infrequently.

The heatmap related to *Back-Office Adjustment Requested* prediction (Figure 3) shows that the attributes related to the type and the reason of bank account closure are influencing the most. When all bank accounts of a customer are closed (labeled by *Closure_Reason=1 - Client lost*) or when the customer decides to close one of its bank accounts among different ones he owns (labeled as *Closure_Reason=2 - Keep bank account. Same dip*), then a *Back-Office Adjustment Requested* is unlikely to happen. This is clearly shown in the heatmap, respectively represented by the values -40374

TABLE II: Online explanations for *Back-Office Adjustment Requested*. Values 1 and 0 indicate if the activity is predicted to occur or not. Explanation followed by (-1): attribute value assigned by the event that precedes the last of respective case.

CASE ID	Back-Office Adjustment Requested	Explanations for Back-Office Adjustment Requested happening	Explanations for Back-Office Adjustment Requested not happening
201810000206	0	-	ACTIVITY=Service closure Request with network responsibility (-2) AND CE_UO=195 (-1)
201811008237	1	CLOSURE_TYPE=Porting	-
201812005701	1	CLOSURE_REASON!=1 - Client lost	-
...	...	...	...

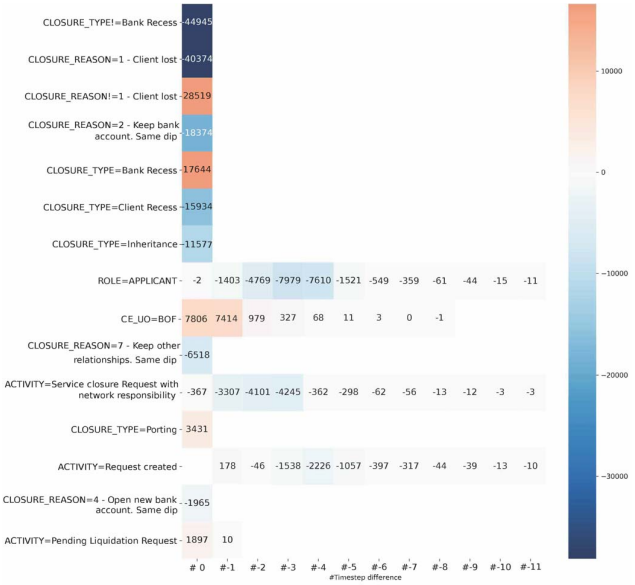


Fig. 3: Offline explanations for *Back-Office Adjustment Requested*

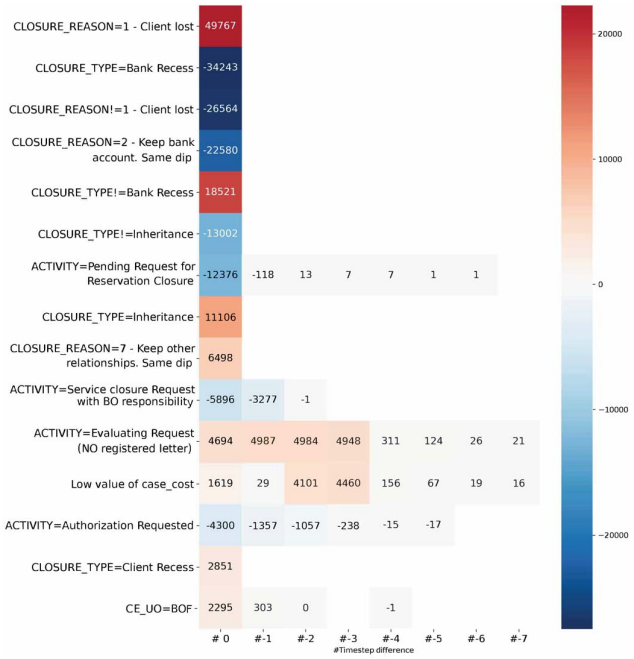


Fig. 4: Offline explanations for *Case cost*

and -18374, which influence is towards not predicting the occurrence of this activity. Values -15934 when the Closure Type is Client Recess (it is the client that decides to close the bank account) and -11577 when it is Inheritance (the bank-account holder is passed away) indicate as well that a *Back-Office Adjustment Requested* is unlikely to happen. Conversely, when the Closure Type is Bank Recess (the bank account is closed by the bank) or it is Porting, then the rework activity *Back-Office Adjustment Requested* is more likely to occur.

Explanations are also used on-line to explain the predictions of running cases. Table II shows the factors that make the model predict whether or not activity *Back-Office Adjustment Requested* is expected to happen for three running cases. Values 1 and 0 indicate that the activity is expected or not to happen, respectively. Let us consider for instance the first case in the table: the rework activity is not expected to happen because two events ago *Service Closure Request with Network Responsibility* has been performed and because the previous event has been performed by the resource 195. Conversely, it is predicted to eventually happen for the other two cases in the table, and the explanation is related to the closure type being Porting and the closure reason not being Client lost.

D. Results on Case Cost prediction

Since this KPI is numerical, we adopted the *Mean Absolute Error*, for which we achieved a value of 0.95 Euros. This is an excellent result, given that the average case cost is 12.86, with standard deviation of 6.41. Figure 4 shows the application for the case cost prediction for the off-line phase. The main factor that contributes to increase the cost of a case is represented by *Closure_Reason=1 - Client Lost*, which is indicated when all bank accounts are going to be closed. The information that the value is positive (i.e. 49767) indicates that the influence is towards increasing the cost. This is mainly caused by the fact that most of the times here the director needs to carefully evaluate the request before proceeding, and the hourly director's wage is certainly higher than that of other bank employees. Nevertheless, this evaluation is not needed when the closure of the bank account is requested by the bank (labeled as *Bank Recess*), therefore the predicted case cost will be smaller (indicated in the heatmap by the negative value -34243). The director is similarly not involved when customers only close one of their bank accounts (*Closure_Reason=2 - Keep bank*

account. Same dip), which is a factor that yields lower costs. Another reason is that when only one between different bank accounts is closed, then of course the process is simpler and less Back-office adjustment activities need to be performed compared to when all bank accounts need to be closed, leading to minor costs. Another indirect evidence that the director's involvement is a factor that increases costs is evident when one looks at the explanations based on *Activity=Evaluating Request (NO registered letter)*. This activity needs a lot of time and is performed by the director, leading to high costs (even higher compared to the case in which a request has only to be authorized). If this activity occurs, the cost will remain permanently high. This is evident in the heatmaps: the fact that this activity has been previously performed is still influencing towards increasing the costs (see columns related to timestep difference -1, -2 and -3, which values are respectively 4987, 4984 and 4948).

VII. Conclusion

A lot of research has been devoted towards increasingly accurate frameworks for predictive process monitoring. Nonetheless, little attention has been paid to ensure that that the resulting predictive-monitoring system is workable in practice. With practical workability, here we intend that the process analysts and stakeholders need to trust the system and its predictions. Previous studies have shown that a necessary condition to build trust is to explain the reason of the provided predictions [13], [4]. Proposals that do not put explanation as a core feature are not going to be adopted in practice.

This paper has put forward a framework to equip predictive-process-monitoring systems with explanations that are intelligible by actors of the process. The framework builds on the most recent state of the art on Explainable AI, and is independent of the actual AI predictive-analytics technique.

However, the operationalization of the framework requires one to select an actual AI technique, and here we opted for predictive models based on LSTM, which the present literature has shown to be the most suitable for the problem in question. The implementation is based on Python, and it has been used for several case studies. Here we reported different KPI predictions for a process run in a financial institute in Italy. The case studies shows that our framework is able to, on the one hand, provide explanations of the most salient features that influence the prediction models and, on the other hand, to provide online explanations on the running cases.

Future work accounts different directions. First, we aim to verify through interviews whether process stakeholders would fully comprehend the heatmaps and the form given to explanations. Second, we aim to explore the possibilities of *Natural Language Generation* techniques to report more user-friendly explanations, instead of the output shown in Tables I and II.

Acknowledgement. This work is supported by MINECO and FEDER funds under grant TIN2017-86727-C2-1-R and by the Department of Mathematics, University of Padua, with HPC resources.

References

[1] Alvarez-Melis, D., Jaakkola, T.S.: On the robustness of interpretability methods. arXiv preprint arXiv:1806.08049 (2018)
[2] Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. In: The 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015)
[3] Breuker, D., Delfmann, P., Matzner, M., Becker, J.: Designing and evaluating an interpretable predictive modeling technique for business processes. In: Business Process Management Workshops. pp. 541–553. Springer (2015)
[4] Doshi-Velez, F., Kim, B.: Towards a rigorous science of interpretable machine learning (2017)
[5] Galanti, R., Coma-Puig, B., de Leoni, M., Carmona, J., Navarin, N.: Explainable predictive process monitoring. arXiv:2008.01807 (2020)
[6] Hochreiter, S., Urgen Schmidhuber, J.: Long Short-Term Memory. Neural Computation **9**(8), 1735–1780 (1997)
[7] Lundberg, S.M., Lee, S.I.: A unified approach to interpreting model predictions. In: Advances in neural information processing systems. pp. 4765–4774 (2017)
[8] Lundberg, S.M., Nair, B., Vavilala, M.S., Horibe, M., Eisses, M.J., Adams, T., Liston, D.E., Low, D.K.W., Newman, S.F., Kim, J., et al.: Explainable machine-learning predictions for the prevention of hypoxaemia during surgery. Nature biomedical engineering **2**(10), 749 (2018)
[9] Márquez-Chamorro, A.E., Resinas, M., Ruiz-Cortés, A.: Predictive monitoring of business processes: A survey. IEEE Transaction on Services Computing **11**(6), 962–977 (2018)
[10] Meacham, S., Isaac, G., Nauck, D., Virginas, B.: Towards Explainable AI: Design and Development for Explanation of Machine Learning Predictions for a Patient Readmittance Medical Application, pp. 939–955 (06 2019)
[11] Molnar, C.: Interpretable Machine Learning (2020)
[12] Navarin, N., Vincenzi, B., Polato, M., Sperduti, A.: LSTM networks for data-aware remaining time prediction of business process instances. In: Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI 2017) (2017)
[13] Nunes, I., Jannach, D.: A systematic review and taxonomy of explanations in decision support and recommender systems. User Modeling and User-Adapted Interaction **27**(3–5), 393–444 (Dec 2017)
[14] Park, G., Song, M.: Prediction-based resource allocation using lstm and minimum cost and maximum flow algorithm. In: International Conference on Process Mining (ICPM). pp. 121–128 (2019)
[15] Rehse, J.R., Mehdiyev, N., Fettke, P.: Towards explainable process predictions for industry 4.0 in the dfki-smart-lego-factory. KI - Künstliche Intelligenz **33**(2), 181–187 (Jun 2019)
[16] Ribeiro, M.T., Singh, S., Guestrin, C.: "why should I trust you": Explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco. pp. 1135–1144 (2016)
[17] Sarthak, J., Wallace, B.C.: Attention is not explanation. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. pp. 3543–3556. Association for Computational Linguistics (2019)
[18] Serrano, S., Smith, N.A.: Is attention interpretable? In: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. pp. 2931–2951. Association for Computational Linguistics, Florence, Italy (2019)
[19] Shapley, L.S.: A value for n-person games. Contributions to the Theory of Games **2**(28), 307–317 (1953)
[20] Shrikumar, A., Greenside, P., Kundaje, A.: Learning important features through propagating activation differences. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. pp. 3145–3153. JMLR. org (2017)
[21] Shu, K., Cui, L., Wang, S., Lee, D., Liu, H.: Defend: Explainable fake news detection. In: International Conference on Knowledge Discovery & Data Mining, SIGKDD. pp. 395–405. ACM (2019)
[22] Sundararajan, M., Taly, A., Yan, Q.: Axiomatic attribution for deep networks. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. pp. 3319–3328. JMLR. org (2017)
[23] Tax, N., Verenich, I., La Rosa, M., Dumas, M.: Predictive business process monitoring with LSTM neural networks. In: Proceedings of 29th International Conference on Advanced Information Systems Engineering (CAiSE 2017). pp. 477–492 (2017)

Design and Evaluation of a Process-aware Recommender System based on Prescriptive Analytics

Massimiliano de Leoni
University of Padua
Padua, Italy
deleoni@math.unipd.it

Marcus Dees
UWV & Eindhoven University of Technology
Amsterdam & Eindhoven, The Netherlands
Marcus.Dees@uwv.nl

Laurens Reulink
CZ
Tilburg, The Netherlands
laurensreulink@gmail.com

***Abstract*—Process-aware Recommender systems (PAR systems) are information systems that aim to monitor process executions, predict their outcome, and recommend effective interventions to reduce the risk of failure. While a PAR system is composed by monitoring, predictive analytics and prescriptive analytics, the lion's share of attention in the recent years has been on the first two, overlooking the last. It seems that process participants are tacitly assumed to take the "right decision" for the most appropriate corrective actions in case of failure's risks. Unfortunately, the assumption of selecting an effective corrective action is not always met in reality. When selecting an intervention, this is mainly based on human judgment, which naturally relies on subjective process' perceptions, instead of objective facts. Experience has shown that, when a fact-based predictive analytics is followed by subjective prescriptive analytics, the positive effect of good predictions are nullified by inconclusive corrective actions, yielding no final improvement. This paper discusses a PAR system that features a data-driven prescriptive analytics framework, which puts aside subjective options and focuses on factual data. The effectiveness of the proposed solution is assessed through the process of a reintegration company, showing a potential increase of customers that find a new job.**

***Index Terms*—Prescriptive Analytics, Recommender Systems, Process Improvement, Machine Learning, Transition Systems**

I. Introduction

Process-aware Recommender systems (hereafter shortened as PAR systems) are a specific class of Information Systems that aim to predict how the executions of process instances are going to evolve in the future, to determine those that have higher chances to not meet desired levels of performance (e.g. costs, deadlines, customer satisfaction) and, consequently, to provide recommendations on which contingency actions should be enacted to try to recover the risky executions. PAR systems are hence expert systems that run in background and continuously monitor the executions, predicting their future and, possibly, recommend intervention actions.

Conceptually, a PAR system is composed by three subsystems/blocks: *(i)* monitoring, which keeps tracks of running process instances, *(ii)* a predictive-analytics block, which forecasts the future outcome of running instances, and *(iii)* a prescriptive-analytics system, which provides recommendations on the running instances that risk to conclude with a poor outcome.

While a large body of research has been conducted on monitoring and predictive analytics, little attention has been paid on generating recommendations (cf. Section V). Process participants are somehow implicitly assumed to take the "right decision" for the most appropriate corrective actions for each case. Unfortunately, the assumption of selecting an effective corrective action is not always met in reality. When selecting an intervention, this is mainly done based on human judgment, which naturally relies on the subjective perception of the process instead of being based on objective facts. In [1], the authors developed a predictive-analytics module that built on machine-learning techniques, and rely on historical data, and discussed potential interventions with process stakeholders. A subsequent field experiment, with real process instances and a selected intervention, showed that, while process instances were predicted rather well, the intervention did not have the desired effect. No significant improvement of the average outcome was observed, even when enacting the selected intervention. The final lesson was that *accurate predictions are crucial, but their effect is nullified if it is not matched by effective recommendations, and effective recommendations must be based on objective evidence from historical process data*.

This paper reports on the design of a prescriptive-analytics technique that recommends which actions/activities to perform next to optimize a certain KPI (Key Performance Indicator) of interest. The recommendations put aside the human subjectivity and rely on the process' transactional data, recorded in the so-called event logs. This way, the recommendations are purely objective, and not biased.

In a nutshell, our prescriptive-analytics proposal relies on a predictive-analytics module. For each running case with predicted, poor KPI values, we first simulate any possible continuation of the running case, i.e. we simulate the situation in which each possible activity is performed as next. Then, we predict the final KPI value for each continuation. Finally, the system recommends the activity (activities) that is (are) predicted to largely improve the KPI value. Of course, one should only simulate those continuations (i.e. next activities) that are meaningful from a domain viewpoint. If one had a

978-1-7281-9832-3/20/$31.00 ©2020 IEEE
DOI 10.1109/ICPM49681.2020.00013

business process model, this would be easy. However, the presence of a process model is a rather strong assumption. Therefore, an event log of complete process instances is used as input to build a transition system that abstracts the observed behavior. Running cases are mapped onto states of this transition system: the meaningful next activities correspond to the transitions enabled at those states.

The three modules (monitoring, predictive and prescriptive analytics) have been implemented as a PAR system in Python. The entire system was assessed on real-life process data of a reintegration company, aiming to maximize the percentage of customers finding a new job (the KPI). The assessment results showed that the use of the PAR system would significantly reduce the percentage of unsuccessful process instances. The statistical significance of the results was also verified, thus providing a sound, successful validation of the quality of our prescriptive analytics.

Section II introduces the basic concept on which our prescriptive analytics builds: event logs, KPIs, and predictive analytics. Section III reports on the design of our framework for prescriptive analytics, while Section IV discusses the implementation and the experiments. Section V compares our prescriptive analytics with the state of the art. Finally, Section VI concludes this paper, summarizing the contribution, the lessons learnt, and the future research directions.

II. Preliminaries

Section II-A introduces the starting point of our process-aware recommender system based on predictive and prescriptive analytics: the event logs. Section II-B formalizes the concept of KPI, while Section II-C introduces preliminary concepts of predictive analytics.

A. Event Logs

The starting point for a prediction system is an *event log*. An event log is a multiset of *traces*. Each trace describes the life-cycle of a particular *process instance* (i.e., a *case*) in terms of the *activities* executed and the process *attributes* that are manipulated.

Definition 1 (Events): Let $\mathcal{A}$ be the set of process' activities. Let $\mathcal{V}$ be the set of process attributes. Let $\mathcal{W}_{\mathcal{V}}$ be a function that assigns a domain $\mathcal{W}_{\mathcal{V}}(x)$ to each process attribute $x \in \mathcal{V}$. Let $\overline{\mathcal{W}} = \cup_{x \in \mathcal{V}} \mathcal{W}_{\mathcal{V}}(x)$. An event is a pair $(a, v) \in \mathcal{A} \times (\mathcal{V} \not\rightarrow \overline{\mathcal{W}})$ where a is the event activity and v is a partial function assigning values to process attributes, with $v(x) \in \mathcal{W}_{\mathcal{V}}(x)$.
A trace is a sequence of events. Note that the same event can potentially occur in different traces, namely attributes are given the same assignment in different traces. This means that potentially the entire same trace can appear multiple times. This motivates why an event log is to be defined as a multiset of traces:[1]

Definition 2 (Traces & Event Logs): Let $\mathcal{E} \subset \mathcal{A} \times (\mathcal{V} \not\rightarrow \overline{\mathcal{W}})$ be the universe of events. A trace σ is a sequence of events, i.e. $\sigma \in \mathcal{E}^*$. An event-log L is a multiset of traces, i.e. $L \subset \mathbb{B}(\mathcal{E}^*)$.

[1]Given a set X, $\mathbb{B}(X)$ indicates the set of all multisets with the elements in X.

Given an event $e = (a, v)$, the remainder uses the following shortcuts: $activity(e) = a$ and $variables(e) = v$. Also, given a trace $\sigma = \langle e_1, \ldots, e_n \rangle$, $prefix(\sigma)$ denotes the set of all σ's prefixes: $\{\langle\rangle, \langle e_1 \rangle, \langle e_1, e_2 \rangle, \ldots, \langle e_1, \ldots, e_n \rangle\}$.

B. Key Performance Indicators

A PAR system aims to optimize the Key Performance Indicators (KPIs) of processes. KPIs can be of any nature:

Definition 3 (KPI Function): Let $\mathcal{E}^*$ be the universe of events defined over a set $\mathcal{V}$ of attributes. A KPI is a function $\mathcal{T} : \mathcal{E}^* \rightarrow \mathbb{R}$ such that, given a complete trace $\sigma \in \mathcal{E}^*$, $\mathcal{T}(\sigma)$ returns the KPI value of σ.

Given a trace $\sigma = \langle e_1, \ldots, e_n \rangle$ that records a complete process execution, the following are four potential KPI definitions:

- **Duration.** $\mathcal{T}_{duration}(\sigma)$ is equal to the difference between the timestamp of e_n and that of e_1.
- **Activity Occurrence.** It measures whether a certain activity is observed in the trace, such as an activity *Open Loan* in a loan-application process. The corresponding KPI definition for the occurrence of an activity A is $\mathcal{T}_{occur\_A}(\sigma)$, which is equal to the number of events in σ that refers to A.
- **Customer Satisfaction.** This is typical KPI to analyze for processes related to service provision. Let us assume, without losing generality, to have a trace $\sigma = \langle e_1, \ldots, e_n \rangle$ where the satisfaction is known at the end, e.g. through a questionnaire. Assuming the satisfaction level is recorded with the last event - say $e_n(sat)$. Then, $\mathcal{T}_{cust\_satisf}(\sigma) = variables(e_n)(sat)$.

C. Predictive Analytics

Predictive Analytics aims to forecast the execution and/or the outcome of a running case. A large share of attention in Process Mining has been devoted to business process predictions (cf. Section V). Within our recommender-system framework for KPI optimization, the predictive analytics block can be defined as follows, specializing the definition by Senderovich et al. [2]. Given a running case $\sigma_{run} = \langle e_1, \ldots, e_k \rangle$, which will eventually end with a trace $\sigma = \langle e_1, \ldots, e_k, e_{k+1}, \ldots, e_n \rangle$, predictive analytics can be abstracted as learning a *predictive-analytics oracle* $\mathcal{P}(\sigma_{run})$ that forecasts the KPI value $\mathcal{T}(\sigma)$. It follows that a perfect prediction is such that $\mathcal{P}(\sigma_{run}) = \mathcal{T}(\sigma_T)$.

Predictive-analytics oracles can be built in several ways, and using several machine-learning approaches. It requires a training event log L_T. A typical predictive-analytics block of a recommender system follows three phases as illustrated in Fig. 1 (cf. [3], [4]):

- **Extract Prefixes.** We build the multiset of all prefixes, which is associated with a corresponding KPI value: $\uplus_{\sigma \in L_T} \uplus_{\sigma' \in prefix(\sigma)} (\sigma', \mathcal{T}(\sigma))$. [2]
- **Encode Prefixes for Training.** The predictor is trained on observation instances (d, i) where d is the vector that

[2]Symbol $\uplus$ indicates the multiset union, namely duplicates are retained.

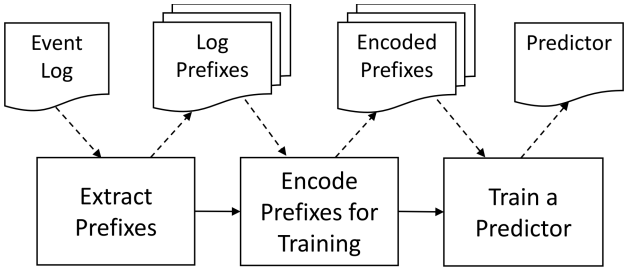


Fig. 1: The training of the predictive model.

encodes the independent variables and i is the dependent variable. The pairs of prefixes and KPI values (σ', κ) need to be converted into an observation instance (d, i) where $i = \kappa$ and d is the encoding of σ'. This encoding can be achieved using consolidated techniques [3], [4]. In particular, Leontjeva et al. show that a frequency-vector encoding is a good balance between abstraction richness and complexity: there is one vector's element per process' activity $a \in \mathcal{A}$, and there is one element per process' attribute $v \in \mathcal{V}$. The value of the element for activity a is equal to number of occurrence of a in σ'; the value for the element of variable v is equal to the value of the latest variable assignment of v.

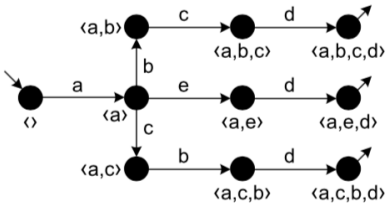
- **Train a Predictor.** The set of encoded prefixes (with the associated KPI values) are the input to train the predictor. A common case is to have prefixes with dozens of different activities and variables, which are encoded via vectors with many elements (see, e.g., the data set reported in Section IV, and used for evaluation). This poses risks of over-fitting due the problem of "curse of dimensionality" [5]. It is thus crucial to perform a *feature selection* to reduce the number of elements [5]. Note that this also reduces the model complexity, and hence it can be learnt and used faster.

III. A Framework for Prescriptive Analytics

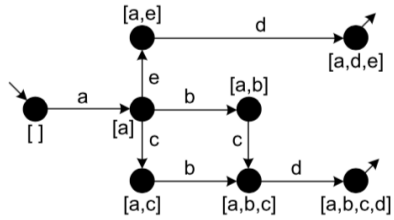
Our prescriptive-analytics framework is based on the availability of a model that was learnt using any of the predictive analytics approaches available in literature (cf. Section V). As indicated in Section II-C, we abstracted from the specific model and approach, assuming that this is exposed as a predictive-analytics function $\mathcal{P}(\sigma)$.

A. Transition System Abstractions of Event Logs

In a nutshell, given a running case, our framework considers all potential ways to continue the case execution, and predicts the risk. However, we do not assume here that process stakeholders need to draw a process model that prescribes how to continue the case executions. This might require a considerable amount of work, and the resulting model might not accurately represent the executions observed in the reality, thus suggesting recommendations that are not valid from a business viewpoint. Following the idea described by van der Aalst et al. [6], [7], valid recommendations are obtained by representing



(a) Using a Sequence Abstraction



(b) Using a multiset abstraction

Fig. 2: Transition systems representing an event log $L = \{\langle a, b, c, d\rangle, \langle a, e, d\rangle, \langle a, c, b, d\rangle\}$ [6]. Event's attributes are abstracted out for simplicity of explanation.

the past executions recorded in the event logs as a transition system. This requires to provide a *state-representation function* $l^{state} : \mathcal{E}^* \to R$ that, for each sequence of events σ, returns a state $l^{state}(\sigma) \in R$ that abstracts σ.

As an example, let us consider an event log $L = \{\langle a, b, c, d\rangle, \langle a, e, d\rangle, \langle a, c, b, d\rangle\}$. We are abstracting here out the event's attributes to keep the example simple. Fig. 2 illustrates two potential transition systems abstracting L: Fig. 2a makes use of a state-representation function $l^{state}(\sigma) = \sigma$, while Fig. 2b uses $l^{state}(\sigma)$ as the multiset of every activity that occurred in σ with the respective cardinality. A transition system is defined as follows:[3]

Definition 4 (Transition-System Abstraction of a Log): Let L be an event log defined over a set $\mathcal{V}$ of attributes, and a set $\mathcal{A}$ of activities. Let $l^{state} : \mathcal{E}^* \to R$ be a state-representation function. A transition system abstracting L is a pair $TS_L = (S, T) \subseteq R \times (R \times \mathcal{E} \times R)$ where

- S = $\cup_{\sigma \in L} \cup_{\sigma' \in prefix(\sigma)} l^{state}(\sigma')$, and
- $T = \{(l^{state}(\sigma'), e, l^{state}(\sigma' \oplus \langle e\rangle))$ s.t. $\exists \sigma \in L$ $\sigma' \oplus \langle e\rangle \in prefix(\sigma)\}$

It is known that transitions systems are generally not suitable to represent business processes, because all possible activity's interleavings need to be explicitly represented, making the transition systems very hard to read in the general case. However, here a transition system is valid as a model, because it is only used internally by the prescriptive-analytics module, and never shown to users.

B. Generating Recommendations

The input of the process prescriptive-analytics system is an event log L and a state representation function l^{state} (cf.

[3]Operator $\oplus$ indicates the concatenation of two sequences.

Def. 4). Log L is used to build a transition-system abstraction $TS_L = (S, T)$, based on l^{state}; L is also employed to build a prediction oracle $\mathcal{P} : \mathcal{E}^* \rightarrow \mathbb{R}$.

Let us consider a trace $\sigma \in \mathcal{E}^*$, which may or may not be part of L. As clarified later in this section, we need to find the set $minDist(TS_L, \sigma) \subseteq S$ of states at minimum distance from σ that can be computed as follows. We first compute the minimum number Υ of changes that are necessary for σ to obtain a trace $\overline{\sigma}$ s.t. $l^{state}(\overline{\sigma}) \in S$. The allowed changes are the primitive supported by the techniques for trace-to-trace alignments [8], namely the insertion and the deletion of events.

Once the minimum number Υ of changes is found, $minDist(TS_L, \sigma)$ is the set of states $l^{state}(\sigma') \in S$ s.t. σ' that is obtained from changing Υ events of σ.

With these concepts at hand, the recommendation for a running case identified by a partial trace $\sigma_{run} \in \mathcal{E}^*$ can be built as follows:

1) Find the set O of transitions (i.e. events) possible by TS_L from the states in $minDist(TS_L, \sigma_{run})$. Namely, $O = \{e \in T : \exists s \in minDist(TS_L, \sigma_{run}), \exists s' \in S.\ (s, e, s') \in T\}$.
2) Return every activity $a \in \mathcal{A}$ that maximizes or minimize $\mathcal{P}$, i.e. such that $\exists e \in O.\ activity(e) = a$, and either $\mathcal{P}(\sigma_{run} \oplus \langle e \rangle) = \max_{e' \in O} \mathcal{P}(\sigma_{run} \oplus \langle e' \rangle)$ or $\mathcal{P}(\sigma_{run} \oplus \langle e \rangle) = \min_{e' \in O} \mathcal{P}(\sigma_{run} \oplus \langle e' \rangle)$.[4]

Here, we assume that every trace in L denotes a legitimate execution of the process, and that L is sufficiently large. In this setting, all transitions enabled in each state of the transition system represent all potential recommendations that make sense from a business viewpoint. Therefore, the set O of enabled transitions at the states $l^{state}(\sigma_{run})$ are in fact activities that may be potentially recommended. Among those in O, some activities are predicted to optimize (i.e., maximize or minimize depending on the KPI) the KPI values if they are performed as the next activities. Those are the activities that the system will recommend.

In practice, activities are typically performed by humans, and their judgement is necessary to carry on process case executions. As a consequence, it might be limiting to only consider those activities that are predicted to optimize the KPI of interest, which are likely one per running case. In fact, our framework extends to the top n activities, namely the n activities whose execution is predicted to lead to the highest KPI values. Value n can be customized at run-time by users (cf. Section IV). In fact, the leitmotif of our framework is that *the recommender system should allow process actors to make informed decisions based on objective facts, but the ultimate choices must remain on the process actors, with their personal judgment*.

IV. Implementation and Evaluation

The framework discussed in Section III has been implemented in Python, and leverages on the machine-learning functionalities of the *scikit-learn* package [5]. The code base is available through GitHub [9].

The remainder of the section reports on the evaluation on a real-life case study. Section IV-A introduces the case study employed for our evaluation, while Section IV-B discusses the construction of the predictive model (i.e. the predictive-analytics oracle). Finally, Section IV-C discusses the setup of the experiments to assess the main contribution of this paper: the prescriptive-analytics technique. Finally, Section IV-D reports on the results of the experiments.

A. The Evaluation Case Study

The evaluation of our approach is performed at a Dutch reintegration company. The objective of the company is to help people who have lost their job to find a new job. Customers of the company are not always capable to find a new job by themselves. Common reasons are their age in combination with having been employed for a long time at the same employer, having been ill, economic circumstances and a mismatch between the type of work that is offered and ones capabilities. The company offers several tools like guiding hints, training, and workshops that can be deployed in an online setting or face-to-face. The company can only provide services to the customer for a limited time. The maximum duration depends on the number of years of work experience of the customer. The minimum number of months of entitlement is 3 and the maximum is 38. A customer can spend less than the maximum duration using the services, e.g., because the customer finds a new job.

The company employs hundreds of professionals that support customers to find a new job. There is a high-level prescriptive process model which prescribes two phases during the customer process. In the first phase, which relates to the first 6 months, a face-to-face contact is required. In the second phase, after the first six months, specific services, taken from the whole set of interventions, need to be offered to the customer. While the interventions that are available are the same throughout the company, it is up to the professional to decide which intervention would benefit a customer the most at each stage of the customer journey.

The initial impression of the company is that the order in which the services are offered can influence the outcome, e.g., having certain types of face-to-face meeting earlier or later in the process might be more effective. Therefore, the PAR system is expected to support the company's employee to choose the right interventions. The system harnesses the knowledge of all professionals and supports every professional in choosing the best next action for a customer.

The KPI that is used in the evaluation is: *the customer found a new job before reaching the maximum service duration*. The KPI is binary and has the value 1 when the customer found a new job before reaching the maximum duration, and 0 otherwise.

The evaluation is based on an event log L consisting of 12296 complete traces with 62 trace attributes and 302 unique

[4] We aim to maximize if higher KPI values are better than lower; otherwise, we aim to minimize.

activities.[5] The attributes refer to properties of the customers (e.g., age). The activities are the interventions by the company employees to support the customer's job seeking, as well as the actions performed by the customers to actively look for a job and to follow-up on the employee's interventions. The trace length is between 1 and 1161 events, with an average of 95 events. For the evaluation, L was divided in three parts:

- log $L_{Training}$ with 6148 traces (50%) was used to train the predictive-analytics oracle $\mathcal{P}$;
- log $L_{Testing}$ with 3074 traces was used to evaluate oracle $\mathcal{P}$, and to simulate the running traces, for which recommendations are created;
- log $L_{Similarity}$ with 3074 traces was used to evaluate the quality of the recommendations.

Under the assumption of no concept drifts, the division is done in a complete random fashion.

B. Evaluation on Predictive Analytics

The predictive analytics module is implemented in accordance with the framework discussed in Section II-C.

In order to build a predictive-analytics oracle $\mathcal{P}$, we used a two-phase approach to choose a suitable predictor model. First, we evaluated three machine-learning techniques: Random Forest, Support Vector Machine, and Decision Tree [10]. We selected the technique that scored the best. This technique was used in the second phase, where the learning parameters were tuned through hyper-parameter optimization.

In both phases, we followed the workflow discussed in Section II-C to train the models. The models were evaluated using two standard metrics: Accuracy, and AUC Score [10]. The AUC score is a valid metric here because the KPI values are not uniformly distributed: most of the executions had good KPI values, namely most of customers have eventually found a job.

Note that the evaluation excluded such Deep-Learning techniques as LSTM networks [11], because those models require a significant training time, while the models that we have employed could already score quite well (see below).

For the first phase, we employed 1000 traces of log $L_{Training}$, which generated 7827 prefixes, which were subsequently encoded. To reduce the number of data-set features, we performed a feature selection, based on *SelectKBest* method [10]. In this first, preliminary phase, Random Forest, Support Vector Machine, and Decision Tree were trained using the default parameters of the respective implementations in the SciPy package for Python.

The results are shown in Table I: Random Forests and Support Vector Machines worked better and equally well. Given the significant difference in training time, we finally opted for Random Forest.

During the second training phase, we employed all traces of log $L_{Training}$, and we performed a model training with hyper-parameter optimization, varying the number of decision trees within the forest, and the number of features to consider when split decision-tree nodes. The best model was obtained with 1000 decision trees, and four decision-tree nodes. To complete the assessment, we used the second log $L_{Testing}$ to measure AUC and accuracy. The best model scored quite well: AUC and accuracy were 0.8145 and 0.8775, respectively.

[5]For reasons of confidentiality, we are not allowed to share the event-log dataset.

Classifier	Accuracy	AUC	Training Time
Decision Tree (DT)	0.657	0.610	**31 ms**
Random Forest (RF)	**0.731**	**0.792**	107 ms
Support Vector Machine (SVC)	0.725	0.734	2580 ms

TABLE I: Metrics Score for the Models Trained by Three Machine-Learning Predictors.

C. Evaluation on Prescriptive Analytics: The Experiment Setup

Here, the prescriptive-analytics module operationalizes the technique discussed in Section III to provide recommendation. The technique builds on a predictive-analytics oracle but remains independent from any specific machine-learning technique employed.

The remainder of this section reports on the evaluation conducted using the oracle that was constructed as illustrated in Section IV-B.

In our experiments, we simulated running cases by considering the 3074 traces in $L_{Testing}$. For each trace $\sigma \in L_{Testing}$, we uniformly extracted a random number k between 1 and $|\sigma| - 1$, and we considered the prefix σ_{run}, obtained considering the first k events of σ.

Denoted with $\mathcal{A}$ as the activities defined in the reintegration process, the evaluation follows the steps below for each trace σ_{run}:

1) We generate one recommendation, namely an activity $a_{\sigma_{run}} \in \mathcal{A}$.
2) We compute the largest set of traces $L_{\sigma_{run}} = \{\sigma_1, \ldots, \sigma_m\} \subset L_{Similarity}$ such that, for each σ_i, there is a prefix of σ_i that is *similar* to σ_{run}.
3) We compute the largest set of traces $L^G_{\sigma_{run}} = \{\sigma^G_1, \ldots, \sigma^G_m\} \subset L_{\sigma_{run}}$ such that, for each σ^G_i, there is a $\sigma^P_i \in prefix(\sigma^G_i)$ that is *similar* to $\sigma_{run} \oplus \langle (a_{\sigma_{run}}, \emptyset) \rangle$, namely when the recommendation $a_{\sigma_{run}}$ was followed.[6] This also defines the set $L^B_{\sigma_{run}} = L_{\sigma_{run}} \setminus L^G_{\sigma_{run}}$ of traces similar to σ_{run} when the recommendation was not followed.
4) We compute the average KPI for the traces in $L^G_{\sigma_{run}}$: $avgKPIRec(\sigma_{run}) = avg_{\sigma \in L^G_{\sigma_{run}}} \mathcal{T}(\sigma)$.
5) We compute the average KPI for the traces in $L^B_{\sigma_{run}}$: $avgKPINoRec(\sigma_{run}) = avg_{\sigma \in L^B_{\sigma_{run}}} \mathcal{T}(\sigma)$.

Note that $avgKPIRec(\sigma_{run})$ and $avgKPINoRec(\sigma_{run})$ are the average when the recommendation is and is not followed, respectively. The average KPI value of the traces in $L^G_{\sigma_{run}}$ and $L^B_{\sigma_{run}}$ aims to simulate the typical scenarios of similar executions with and without recommendation. Clearly, the definitive assessment would be to test the recommender system

[6]Symbol $\emptyset$ denotes the function with empty domain.

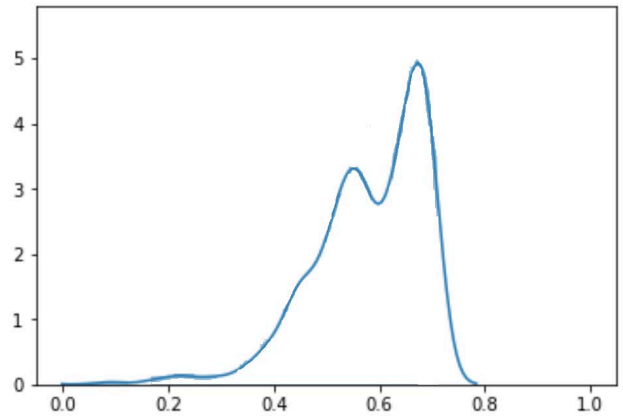


Fig. 3: Distribution of KPI values when recommendations were not followed (Average 0.63). The y axis represents the number of cases.

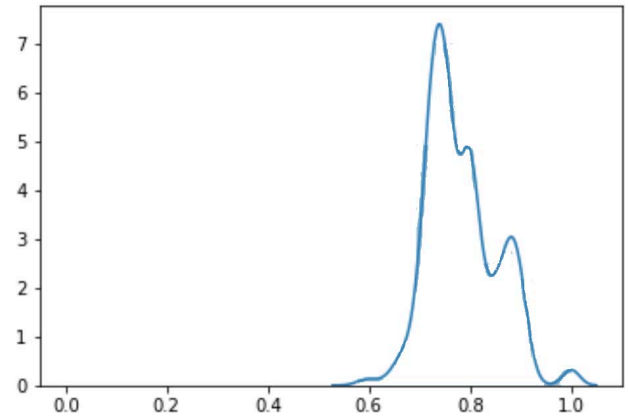


Fig. 4: Distribution of KPI values when recommendations were followed (Average 0.74). The y axis represents the number of cases.

	Traces	**Average Followed**	**Average Not Followed**	Δ
Improvement	1205	0.78	0.58	0.20
Worsening	137	0.56	0.64	-0.08

TABLE II: Number of process instances for which the recommendations led to a statistically significant improvement or worsening of KPI values.

in a real production environment, but the company did not allow this because of the potential consequences on their business.

The above procedure requires one to find the similar traces in $L_{Similarity}$ to a given trace σ_{run}. To achieve this, the prefixes of each trace $L_{Similarity}$ are divided in buckets. A first bucket contains the prefixes with duration of up to one month, where the duration is the difference between the timestamp of the last event and that of the first event of the prefix. A second bucket contains the prefix with duration between one and two months, the third bucket between two and three months, and so on. The prefixes of each bucket are encoded as vectors, analogously to the procedure that was used to learn the prediction oracle (cf. point "Encode Prefix for Training" in Section II-C). Then, a set of clusters is created for each bucket, using Agglomerative Hierarchical clustering along with a principal component analysis to reduce the number of elements of the vectors [10]. The Silhouette score was used to ensure a good quality, while each cluster contained at least 30 traces (ca. 1% of the traces in $L_{Similarity}$). The use of hierarchical clustering has made it possible to set the 30-trace constraint, which motivates the choice of the clustering algorithm itself. Then, we determine the bucket b to which σ_{run} would belong. Finally we encode σ_{run} as vector ν, which ultimately is associated with the b's cluster whose centroid is closer to ν. This centroid contains the prefixes of the traces that are similar to σ_{run}.

It is worthwhile noting that the use of clustering is not part of our prescriptive-analytics framework. It is only used in the case study to determine the traces in a certain log that are similar to a given one.

In sum, given the log $L_{Testing} = \{\sigma^1, \ldots, \sigma^m\}$, we simulated a log of running cases $L_{run} = \{\sigma^1_{run}, \ldots, \sigma^m_{run}\}$. where σ^i_{run} contains the first k events, with k randomly chosen between 1 and $|\sigma^i|-1$. Then, we compute $avgKPIRec(\sigma^i_{run})$ and $avgKPINoRec(\sigma^i_{run})$, for all $1 \leq i \leq m$.

Section IV-D reports on the results obtained through the experimental setup discussed here.

D. Evaluation on Prescriptive Analytics: The Results

Fig. 3 shows the distribution of $avgKPINoRec(\sigma^i_{run})$ for all $\sigma^i_{run} \in L_{run}$. This corresponds to the distribution of KPI values when recommendation is not followed. Recall that, in our case study, a KPI with the value of 1 is the highest value, while 0 is the lowest. Fig. 4 shows the distribution of $avgKPIRec(\sigma^i_{run})$ for all $\sigma^i_{run} \in L_{run}$, namely when recommendations are followed. By comparing Fig. 3 and Fig. 4, one can easily observe how the distribution is certainly closer to 1, when recommendations are indeed followed. The distribution's average indeed increased from 0.63 to 0.74.

The analysis of the distribution of KPI values is certainly valid to gain a general insight into the quality of recommendation. However, this does not directly say how many traces are observing a statistically significant improvement of KPI values when recommendations are followed.

Recall that, for each running trace σ^i_{run}, $L^G_{\sigma^i_{run}}$ and $L^B_{\sigma^i_{run}}$ are the traces similar to σ^i_{run} when the recommendation is or is not followed, respectively. While Fig. 3 and Fig. 4 give a qualitative indication of the presence of a KPI improvement when recommendations are followed, a definitive confirmation requires one to count the number of traces σ^i_{run} for which the average KPI value of traces in $L^G_{\sigma^i_{run}}$ is significantly higher or lower than those in $L^B_{\sigma^i_{run}}$, from a statistical perspective. For each trace σ^i_{run}, we conducted a Z Test to compare the average KPI value of the traces in $L^G_{\sigma^i_{run}}$ with the average of the traces

in $L^B_{\sigma^i_{run}}$. The results are shown in Table II, and illustrates that, when recommendations are followed, there is a significant improvement of KPI values for 1205 out of 3074 traces (ca. 39%), while recommendations had a negative KPI impact on 137 traces (ca. 4%). Columns *Average Followed* and *Average Not Followed* illustrate the KPI values when recommendations were followed or not. For those recommendations with a significant improvement, the improvement was ca. 0.2 when the recommendations were followed. Conversely, for those with a significant worsening of KPI values, the negative effective was more limited: around 0.08. This ultimately means that, when executions followed the recommendations, *the KPI value significantly increased for 39% of the executions*, and the recommendations were counter-productive for few executions, just a limited 4%.

The cases with a significant KPI improvement received recommendations at different moments in time, as shown in Fig. 5. The x axis of the figure represents the time in months, and the y axis indicates the amount of KPI improvement. A point (x, y) of the graph indicates that the traces that received recommendation after x months had an average KPI improvement of y. The curve is obtained via interpolation of discrete points. Fig. 5 highlights that almost every process execution with a significant KPI improvement received a recommendation not later than four months since the execution started. Note that, if one considers every trace of the initial event log of 12,296 traces (cf. Section IV-A), the average case duration is around 12 months with standard deviation of 10.66 months. The average process-instance duration is hence significantly higher than the four months shown in Fig. 5. This shows that *recommendations are almost only useful when followed at the beginning of the case*. From a business viewpoint, this means that, if a customer is at high risk to not find a job, any supporting intervention (i.e. recommendation) should be put in place as soon as possible to have higher chance of success.

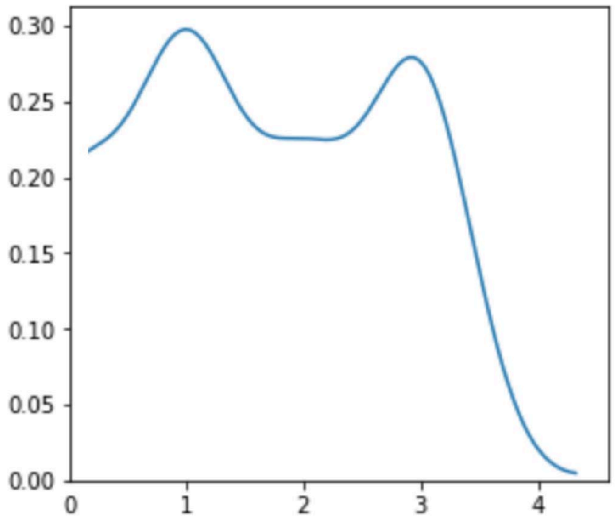


Fig. 5: The relation between the earliness of the recommendation and the amount of KPI improvement when it is followed. The x and y axes respectively represent the time in months when the recommendation was given and followed and the amount of improvement.

V. Related Works

The realm of Process Mining has proposed several research works on Predictive Analytics [3], [12], [13]. Conversely, *"little attention has been given to providing recommendation"* and *"there is still work to be done in this direction [i.e. process-aware recommender systems]"* [12].

In [14], Conforti et al. discusses a recommender system that prioritizes the activities that are offered for executions, without reasoning on additional activities that can be beneficial to put risky process instances back into the right track. To avoid violations of constraints, Maggi et al. [15] aim to provide personalized recommendations to process participants on which activity to work on as next among those in queue for execution. Rozinat et al. propose a framework based on simulation, but they do not provide a concrete operationalization [16]. The same lack of concrete operationalization is also observed in a number of architectures for Process-aware Recommender systems (a.k.a. Operational Support), such as that by Maggi and Westergaard [17]: the infrastructure is only intended to allow researchers to build and deploy their own predictive- and/or prescriptive-analytics algorithms.

Schonenberg et al. [18] and Schobel and Reichert [19] are among the few approaches that propose concrete implementations of prescriptive-analytics modules for recommender systems. However, both approaches base their recommendation on the KPI's average in similar traces. These approaches are certainly valuable when there are sufficiently many traces that are largely the same. However, first, they only focus on the activity information attached to the events, instead of the full payload, as our approach does. Second, they cannot generalize predictive patterns (e.g., on traces that are not very similar), which conversely our framework based on machine-learning is able to do. An empirical evidence of this is in [20]: this paper illustrates that machine-learning techniques are able to provide more accurate predictions (and hence recommendations) with respect to just considering averages.

A very recent proposal of prescriptive analytics based on machine learning is by Weinzierl et al. [21]. However, this proposal is only able to recommend for reducing the remaining time, while we aim at a generic, user-customizable KPI. The prescriptive analytics proposed in this paper can be potentially integrated with that by Teinemaa et al. [22], which helps better prioritize the process cases that require attention, and recommendations.

VI. Conclusions

Process-aware Recommender systems are a new breed of Information Systems that aim to monitor process executions, predict their final outcome and, when the latter is unsatisfactory, suggests corrective actions, in the form of recommen-

dations about the activities to be executed next, to try to get them back on track. While a large share of research has been focused on monitoring and predicting, the aspects related to recommending have been insufficiently considered. It seems that process stakeholders have been assumed to be able to recover from those executions on the basis of their subjective judgment. A few experience reports illustrated that this is not the case: recommendations need to be based on evidence from historical data.

This paper puts forward a prescriptive-analytics framework that relies on process event data. It relies on machine-learning techniques to be able to *(i)* discover the execution patterns, *(ii)* generalize them, and *(iii)* correlate them with KPI values.

By combining the prescriptive analytics with monitoring and predictive analytics, we have been able to design a fully-fledged Process-aware Recommender system.

The framework has been implemented in Python, and evaluated with real-life event data from a Dutch reintegration company. The results illustrate that ca. 39% of the analyzed process executions would have a statistically significant improvement of the KPI values if the personalized recommendations were followed. Recommendations showed to be counter-productive for just 4% of the executions.

We acknowledge that the validity of the evaluation is threatened by the fact that the conclusions have been drawn on the basis of the comparison with similar traces. Conversely, a definitive assessment would require to deploy the system in a production environment, and use it to provide support to a certain fraction of the running cases. At the same time, the rest of the cases would continue as before, without the system. This would allow us to compare the KPI values with and without recommendation. Although the on-the-field experimentation was the initial intention of this research work, it showed itself to be a goal that is hard to achieve. Companies are extremely reluctant to carry on this sort of experimentation, because it can hamper their own business. This is also due to the fact that the trust in the recommender system can only be built if recommendations are coupled with human intelligible explanations that motivate them [23].

The considerations above delineate some directions worthwhile exploring: on the one hand, we aim to test our recommender system in a production environment; on the other hand, we want to equip it with explanations of the recommendations.

Last but not least, our prescriptive analytics only focuses on activities, event and trace attributes, and does not consider the aspects related to the time between events, such as the process-instance duration (time between the first and last activity), or the time elapsed since the last performed activity. We plan to incorporate these time-related aspects, which might additional ingredients towards more accurate recommendations.

REFERENCES

[1] M. Dees, M. de Leoni, W. M. P. van der Aalst, and H. A. Reijers, "What if process predictions are not followed by good recommendations?" in *Proceedings of the Industry Forum at BPM 2019*, ser. CEUR Workshop Proceedings, vol. 2428. CEUR-WS.org, 2019, pp. 61–72.

[2] 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," *Lecture Notes in Computer Science Business Process Management*, p. 306–323, 2017.

[3] I. Teinemaa, M. Dumas, M. L. Rosa, and F. M. Maggi, "Outcome-oriented predictive process monitoring: Review and benchmark," *ACM Trans. Knowl. Discov. Data*, vol. 13, no. 2, Mar. 2019.

[4] A. Leontjeva, R. Conforti, C. Di Francescomarino, M. Dumas, and F. M. Maggi, "Complex symbolic sequence encodings for predictive monitoring of business processes," in *International Conference on Business Process Management*. Springer, 2015, pp. 297–313.

[5] A. C. Müller and S. Guido, *Introduction to Machine Learning with Python: A Guide for Data Scientists*. OReilly, 2017.

[6] W. M. P. van der Aalst, *Process Mining: Data Science in Action*. Berlin: Springer-Verlag, 2011.

[7] W. M. P. van der Aalst, M. Pesic, and M. Song, "Beyond process mining: From the past to present and future," in *Advanced Information Systems Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 38–52.

[8] R. P. J. C. Bose and W. M. P. van der Aalst, "Trace alignment in process mining: Opportunities for process diagnostics," in *Proceedings of the 8th International Conference on Business Process Management*, ser. BPM'10. Berlin, Heidelberg: Springer-Verlag, 2010, p. 227–242.

[9] L. T. W. Reulink, "Koala," GitHub. [Online]. Available: https://github.com/laurensreulink/koala

[10] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[11] N. Tax, I. Verenich, M. La Rosa, and M. Dumas, "Predictive business process monitoring with LSTM neural networks," in *Proceedings of 29th International Conference on Advanced Information Systems Engineering (CAiSE 2017)*, 2017, pp. 477–492.

[12] 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, 2018.

[13] M. Camargo, M. Dumas, and O. González-Rojas, "Learning accurate lstm models of business processes," in *Business Process Management*, T. Hildebrandt, B. F. van Dongen, M. Röglinger, and J. Mendling, Eds. Cham: Springer International Publishing, 2019, pp. 286–302.

[14] R. Contorti, M. de Leoni, M. La Rosa, W. M. P. van der Aalst, and A. H. M. ter Hostede, "A recommendation system for predicting risks across multiple business process instances," *Decision and Support System*, vol. 69, no. C, p. 1–19, Jan. 2015.

[15] F. M. Maggi, C. Di Francescomarino, M. Dumas, and C. Ghidini, "Predictive monitoring of business processes," in *Advanced Information Systems Engineering*. Springer, 2014, pp. 457–472.

[16] A. Rozinat, M. T. Wynn, W. M. P. van der Aalst, A. H. M. ter Hofstede, and C. J. Fidge, "Workflow simulation for operational decision support," *Data Knowledge Engineering*, vol. 68, no. 9, p. 834–850, Sep. 2009.

[17] F. M. Maggi and M. Westergaard, "Designing software for operational decision support through coloured petri nets," *Enterprise Information System*, vol. 11, no. 5, p. 576–596, 2017.

[18] H. Schonenberg, B. Weber, B. van Dongen, and W. van der Aalst, "Supporting flexible processes through recommendations based on history," in *Business Process Management*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 51–66.

[19] J. Schobel and M. Reichert, "A predictive approach enabling process execution recommendations," in *Advances in Intelligent Process-Aware Information Systems: Concepts, Methods, and Technologies*, G. Grambow, R. Oberhauser, and M. Reichert, Eds., 2017, pp. 155–170.

[20] M. Polato, A. Sperduti, A. Burattin, and M. de Leoni, "Time and activity sequence prediction of business process instances," *Computing*, vol. 100, no. 9, pp. 1005–1031, 2018.

[21] S. Weinzierl, S. Zilker, M. Stierle, G. Park, and M. Matzner, "From predictive to prescriptive process monitoring: Recommending the next best actions instead of calculating the next most likely events," in *Proceedings of the 15th International Conference on Wirtschaftsinformatik*, 2020.

[22] I. Teinemaa, N. Tax, M. de Leoni, M. Dumas, and F. M. Maggi, "Alarm-based prescriptive process monitoring," in *Proceedings of Business Process Management Forum - BPM Forum 2018*, ser. Lecture Notes in Business Information Processing, vol. 329. Springer, 2018, pp. 91–107.

[23] C. Molnar, *Interpretable Machine Learning*, 2020, available at https://christophm.github.io/interpretable-ml-book/.

Detecting System-Level Behavior Leading To Dynamic Bottlenecks

Zahra Toosinezhad*, Dirk Fahland*, Özge Köroğlu*, Wil M.P. van der Aalst†*
*Eindhoven University of Technology, Eindhoven, The Netherlands, email: z.toosinezhad@tue.nl, d.fahland@tue.nl
†RWTH Aachen, Department of Computer Science, Aachen, Germany, wvdaalst@pads.rwth-aachen.de

***Abstract*—Dynamic bottlenecks occur when some cases in a particular part of the process are temporarily delayed. In performance-optimized systems such as production systems, warehouse automation systems, and baggage handling systems, such bottlenecks are rare, bounded in time and location, but costly when they occur and propagate through the system. Detecting and understanding the situations that cause such bottlenecks is crucial for mitigating and preventing processing delays. Classical process mining techniques that analyze performance along individual cases cannot detect these phenomena and their causes. We show that undesired system-level behavior can be detected when identifying temporal event patterns across different cases in the same process step. Conceptualizing these patterns as system-level events allows us to correlate them into cascades of system-level behavior using spatio-temporal conditions. We discover classes of frequent patterns in these cascades that describe behaviors that precede bottlenecks. Applied on event data of a major European airport, our approach could fully automatically detect cascades of undesired system-level behavior leading to dynamic bottlenecks. Each detected cascade was verified as a correct causal explanation for a dynamic bottleneck due to the physical system layout and its processing.**

***Index Terms*—Process mining, Performance analysis, Dynamic bottlenecks, Event aggregation, Event correlation**

I. Introduction

A primary objective of applying process mining is detecting and understanding process performance problems to prevent unnecessary delays or *bottlenecks* through process redesign or early detection and mitigation. Bottlenecks are observed when cases or items are processed too slow, leading to an accumulation of unprocessed work. Bottlenecks are caused by the unavailability of a worker or machine or when work volume exceeds work capacity, resulting in growing queues and longer waiting times [1]. The most common approach for identifying bottlenecks from event logs is to calculate for each case or item the time difference between any two subsequent process steps [2]. Aggregating these time differences over all cases reveals which process steps take longer than average, indicating a *"static" bottleneck* observed for the majority of the cases due to structural capacity problems.

A. Dynamic Bottlenecks

Performance-optimized processes in production systems, warehouse automation systems, and Baggage Handling Systems (BHS) already established sufficient processing capacity to avoid static bottlenecks, often through a lean design process [3]. Here, bottlenecks only arise *dynamically* when the process operates outside its standard operating conditions, such as work volume temporarily exceeding expected capacity or processing capacity temporarily falling below intended levels. Fig. 1 illustrates an example from a BHS. We observe a higher load of cases (bags) $4, 5, 6$ between $L1$ and $L2$ that can still be processed normally with additional cases 3 and 7 about to enter. This high load propagates forward via conveyor belts to $L2, L3, L4$. Between $L4$ and $L5$ cases $2, 4, 3, 5$ saturate the physical system capacity. Processing at $L4$ is temporarily slowed down by stopping the conveyor belt from $L3$ to $L4$, and cases $6, 7$ are *blocked* for 2 minutes.

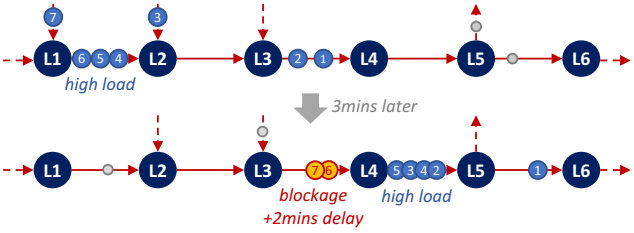


Fig. 1: Formation of a dynamic bottleneck.

Such *dynamic bottlenecks* are rare, bounded in time and location, but costly when they occur. In the BHS example, a delay of 2 minutes could mean bags $6, 7$ might miss a scheduled flight. Bottlenecks may propagate through the system similar to a traffic jam: processing at $L3$ in Fig. 1 may slow down as well, causing further bottlenecks before $L3$. Understanding under which circumstances dynamic bottlenecks emerge and propagate (and how far) would enable process redesign or the development of suitable early-warning prediction models enabling human or automated intervention.

B. Research Problem

In this paper, we consider the *research problem of (1) detecting dynamic bottlenecks and (2) detecting the process behavior which frequently precedes dynamic bottlenecks from basic process event logs*. The process behavior leading to a bottleneck *cannot* be described in terms of a single case or group of cases: $\{4, 5, 6\}$ between $L1$ and $L2$ form the initial high-load situation, but $\{2, 3, 4, 5\}$ between $L4$ and $L5$ form the later high-load situation causing the blocking of $\{6, 7\}$ before $L4$. If the bottleneck propagates further towards $L2$, completely different cases will be affected. As a result *case-oriented process mining techniques are unable to detect dynamic bottlenecks and how they form and propagate*.

978-1-7281-9832-3/20/$31.00 ©2020 IEEE
DOI 10.1109/ICPM49681.2020.00014

C. Approach

We argue that solving the above problems requires studying "system-level" behavior which can only be observed indirectly from case-level events (of different cases) that have a specific characteristic in a specific part of the process, e.g., events related to bags $\{4, 5, 6\}$ between $L1$ and $L2$ forming a "high-load" dynamic. From this angle, the following three sub-problems arise which we propose to solve as follows.

(1) *Automatically detect "outlier" system-level behaviors that are different from normal operations and bounded in location and in time, i.e., the specific process steps and time interval where the process operated outside its regular parameters.* Formally, given an event log L of case-level events E characterized by activity, time, and case identifier, identify all sub-sets $s_1, ..., s_N \subseteq E$ of events of L whose characteristics significantly deviate from other sets of events, e.g., higher load and dynamic bottlenecks. The subsets s_i may overlap if they represent different overlapping characteristics. In this paper, we conceptualize the problem in a general way and propose to aggregate each set s_i into a *system-level event* for further analysis; we provide efficient techniques to detect time-bounded high-load situations and dynamic bottlenecks in performance-optimized systems such as a BHS.

(2) *Automatically identify those system-level events which are meaningfully* correlated *to each other and can be considered as a cascade of causally related system-level events.* This necessitates the inclusion of domain-knowledge. In this paper, we conceptualize the problem in terms of temporal and spatial distance properties between system-level events. We then identify correlated events by *querying* for system-level events with low temporal and spatial distance, i.e., dynamics in adjacent process steps that overlap in time. We show that the correlated, viz. queried, system-level events describe a partially-ordered system-level behavior that we call a *cascade*.

(3) *Automatically identify frequent patterns of correlated system-level events in system-level cascades that result in dynamic bottlenecks.* We show that applying existing sub-graph mining algorithms can efficiently detect such frequent patterns in cascades.

D. Results

We applied the above techniques of system-level event detection, querying, and grouping into system-level cascades, and frequent pattern detection on the complete baggage-level event data of 7 days of a BHS of a major European airport. We identified 1029 instances of dynamic bottlenecks and 123187 instances of other outlier behaviors. We identified four basic types of frequent patterns that precede these bottlenecks; all identified patterns were confirmed as correctly describing cause-effect behavior for forming dynamic bottlenecks.

In the following, we discuss related work in Sect. II. We then introduce an event model that considers events at different levels of abstraction in Sect. III to structure our research problem. We discuss how to detect system-level events in Sect. IV, how to detect cascades of correlated system-level events in Sect. V, and how to find patterns in cascades in Sect. VI. We report on our results on synthetic and real-life event logs in Sect. VII and discuss our findings and future work in Sect. VIII.

II. Related work

The primary approach for *performance outlier analysis in process mining* is to aggregate time differences between any two process steps and identify outlier values [2]. However, these techniques can only identify static bottlenecks affecting all (or most) cases. Event interval analysis [4] allows analyzing durations between any two process steps on a more fine-grained level based on event and case attributes, but cannot capture patterns involving multiple cases. The visual analytics technique of the Performance Spectrum reveals time-bounded performance patterns [5] over multiple process steps. These performance patterns can be understood as emergent system-level behavior formed by a group of cases in a specific time-window, such as batching [6]. We contribute techniques to detect performance patterns of multiple cases (system-level events) with a negative impact on performance, esp. bottlenecks, and how they propagate in time and to other process steps (as system-level dynamic).

Behavioral outlier analysis techniques in process mining identify infrequent event sequences [7] or infrequent event contexts [8] in a case of correlated events. The problem of this paper is to identify frequent patterns over emergent system-level events that are initially uncorrelated.

Techniques for finding sequential patterns in spatio-temporal event data [9] are inapplicable due to the distributed nature of processes. Process event correlation techniques either use case-level attributes to correlate events [10] or a process model describing the expected behavior [11]. This information is not available for emergent system-level events. By abstracting a process as a sequence of queues and visualizing the queue parameters over time, Staged Process Flows [12] allow visually analyzing how performance problems propagate through a process and affect multiple cases. This technique however assumes a sequential process and outlier behavior is not captured as a data feature for further analysis.

Queueing concepts are also used to discover congestion graphs [13] which model workload and resource availability over time in a Markov state for improved remaining time prediction. The congestion graph model is local to one process step and does not consider how performance problems may propagate. Our work aims at detecting, for instance, situations of high workload and resource unavailability and how they propagate to other process steps, thus providing a large description of performance dynamics.

Outside process mining, studies in *distributed system failure analysis* [14], [15] revealed the importance of analyzing correlations of outlier behavior and failures to increase resilience and performance in IT systems [16] and in IoT environments [17]. No existing work studies cascades of correlated failures for frequent patterns. Various works address finding outliers and their causal relations in *traffic data streams*. Outliers are identified through temporal properties in a specific spatial area

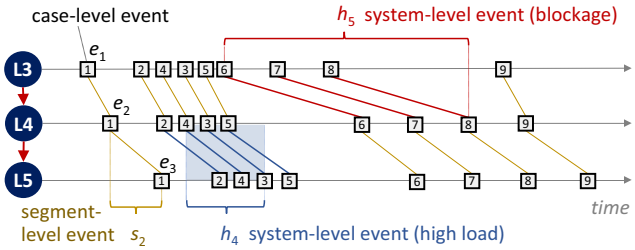


Fig. 2: Case, segment, and system-level events.

[18]–[20]. Outliers of a single type are correlated by building trees based on spatial and temporal properties of outliers [18] or by training a deep neural network which also allows short-term prediction [19]. Spatio-temporal correlations between different outliers can be learned through an attention network [20]. However, these works do not distinguish different types of outlier behavior. These techniques cannot visualize and explain longer cascades of multiple outlier events over time and their frequencies.

III. Event Definitions on Different Levels

To precisely formulate the three research questions of Sect. I, we introduce a conceptual event model having 3 levels. *Case-level* events records updates to a case as usual; *segment-level* events capture behavioral properties of pairs of directly related events; *system-level events* describe sets of consecutive segment-level events with specific properties. Figure 2 illustrates all 3 event notions for the behavior described in Fig. 1.

We assume our input events to be records of actions that happened to a specific case or object at a specific point in time, i.e., the standard event notion in process mining. We call a relation $< \subseteq E \times E$ an *acyclic* order iff its transitive closure $<^*$ is irreflexive, i.e., $<$ has no (self-)cycle.

Definition 1 (Case-level event log)**.** *A* case-level event log $L^c = (E^c, \#^c, <^c)$ *is a set* E^c *of events;* $\#^c$ *defines attributes such as case id* $\#^c_{id}(e)$*, activity* $\#^c_{act}(e)$*, and time* $\#^c_{time}(e)$ *for each* $e \in E^c$*; events are ordered by an acyclic order wrt. case id and time* $<^c \subseteq \{(e_1, e_2) \in E^c \times E^c \mid \#^c_{id}(e_1) = \#^c_{id}(e_2) \wedge \#^c_{time}(e_1) \leq \#^c_{id}(e_2)\}$.

Note that $<^c$ orders only events in the same case and according to time, but $<^c$ can be stricter than a partial order. In the following, we assume that $<^c$ describes "directly causally precedes" which can be derived using binary token flows [21], transitive reduction of a partial order [8], or may simply hold in the data, e.g., a bag moving on a physical conveyor belt. In Fig. 2, each square shows a case-level event, each diagonal line between two case-level events shows $<^c$. Tab. I shows a part of a case-level event log for our running example of Fig. 1 (*not* visualized in Fig. 2); $e_{15} <^c e_{18}$ and $e_{18} <^c e_{22}$ but $e_{15} \not<^c e_{22}$.

The set of activities in L^c is $\Sigma = \{\#^c_{act}(e) \mid e \in E^c\}$. We call a pair $(a, b) \in \Sigma \times \Sigma$ a *segment* of the process describing the passage from activity a to the next activity b [5]. Two

TABLE I: Case-level events of running example

	id	*act*	*time*
...	...	...	...
e_{10}	1	L4	10:21:30
e_{11}	1	L5	10:21:50
e_{12}	2	L4	10:23:10
e_{13}	2	L5	10:23:30
e_{14}	4	L4	10:23:25
e_{15}	5	L3	10:23:40
e_{16}	4	L5	10:23:45
e_{17}	3	L4	10:23:40
e_{18}	5	L4	10:23:55
...	...	...	...

...	...	...	...
e_{19}	3	L5	10:24:00
e_{20}	6	L3	10:23:55
e_{21}	7	L3	10:24:30
e_{22}	5	L5	10:24:05
e_{23}	6	L4	10:26:00
e_{24}	8	L3	10:24:40
e_{25}	7	L4	10:26:15
e_{26}	6	L5	10:26:20
e_{27}	8	L4	10:26:20

case-level events $e_1 <^c e_2$ with $\#^c_{act}(e_1) = a, \#^c_{act}(e_2) = b$ describe that the cases or objects $\#^c_{id}(e_1)$ and $\#^c_{id}(e_2)$ passed through segment (a, b). We capture this behavioral observation as an aggregate event, called a *segment-level event* as shown in Fig. 2. The segment-level events are fully defined by the case-level event log L^c as follows.

Definition 2 (Segment-level log)**.** *Let* $L^c = (E^c, \#^c, <^c)$ *be a case-level event log. A* segment-level log $L^s = (E^s, \#^s, <^s)$ *of* L^c *has events* $E^s = <^c$ *(the causally related events of* L^c*) where each* segment-level *event* $s = (e_1, e_2) \in E^s$ *has case id* $\#^s_{id}(s) = \#^c_{id}(e_1) = \#^c_{id}(e_2)$*, source and target activity* $\#^s_{src}(s) = \#^c_{act}(e_1)$, $\#^s_{tgt}(s) = \#^c_{act}(e_2)$*, start and end time* $\#^s_{start}(s) = \#^c_{time}(e_1)$, $\#^s_{end}(s) = \#^c_{time}(e_2)$*. Events are acyclically ordered wrt. segment and time by* $<^s \subseteq \{(s_1, s_2) \in E^s \times E^s \mid \#^s_{src}(s_1) = \#^s_{src}(s_2) \wedge \#^s_{tgt}(s_1) = \#^s_{tgt}(s_2) \wedge (\#^s_{start}(s_1) < \#^s_{start}(s_2) \vee \#^s_{end}(s_1) < \#^s_{end}(s_2))\}$.

Each diagonal line between two case-level events in Fig. 2 is a segment-level event. Table II shows the segment-level event log for the case-level events in Tab. I; for instance $s_{75} = (e_{15}, e_{18})$ and $s_{77} = (e_{18}, e_{22})$. Each segment-level event defines a spatio-temporal interval from $\#^s_{src}(s)$ to $\#^s_{tgt}(s)$ during time $[\#^s_{start}(s); \#^s_{end}(s)]$.

In contrast to case-level events, $<^s$ orders the segment-level events of cases per segment, e.g., in Tab. II $s_{75} <^s s_{78}$ (for cases 5 and 6 in $(L3, L4)$). Def. 2 requires that the order $<^s$ is consistent with at least start or end time. Though, $<^s$ can be stricter than the temporal order (as in Def. 1). In the following, we assume $s_1 <^s s_2$ iff s_1 *directly starts before* s_2, i.e., their start events are neighbors on the time-axis of their start activity in Fig. 2. We write $E^s(a, b)$ for events $s \in E_s$ with $\#^s_{src}(s) = a$ and $\#^s_{tgt}(s) = b$ in segment (a, b).

Fig. 2 shows how a set of consecutive segment-level events may form a behavior that we can recognize as a pattern at the system level. For instance, the segment-level events of cases $2, 3, 4, 5$ in $(L4, L5)$ have a shorter arrival rate than other segment-level events causing a short interval of *high-load*. The segment-level events of cases $6, 7, 8$ in $(L3, L4)$ have a longer duration causing a *dynamic bottleneck*.

To be able to reason about how the high-load interval relates to the bottleneck, we aggregate a set of consecutive segment-level events into a *system-level event*.

TABLE II: Segment-level event log

	id	*src*	*tgt*	*start*	*end*
s_{72}	1	L4	L5	10:21:30	10:21:50
s_{73}	2	L4	L5	10:23:10	10:23:30
s_{74}	4	L4	L5	10:23:25	10:23:45
s_{75}	5	L3	L4	10:23:40	10:23:55
s_{76}	3	L4	L5	10:23:40	10:24:00
s_{77}	5	L4	L5	10:23:55	10:24:05
s_{78}	6	L3	L4	10:23:55	10:26:00
s_{79}	7	L3	L4	10:24:30	10:26:15
s_{80}	8	L3	L4	10:24:40	10:26:20
s_{81}	6	L4	L5	10:26:00	10:26:20

Definition 3 (System-level event log)**.** *Let L^s be a segment-level event log. A* system-level event log $L^h = (E^h, \#^h, <^h)$ *(over L^s) defines a set of system-level events E^h where for each $h \in E^h$, $\#^h_{type}(h)$ is the event type, $\#^h_{src}(h)$ and $\#^h_{tgt}(h)$ are source and target, $\#^h_{sev}(h) \subseteq E^s(\#^h_{src}(h), \#^h_{tgt}(h))$ are segment-level events from L^s, $\#^h_{start}(h) = \min_{s \in \#^h_{sev}(h)} \#^s_{start}(s)$ is the start time and $\#^h_{end}(h) = \max_{s \in \#^h_{sev}(h)} \#^s_{end}(s)$ is the end time. Event $h \in E^h$ is* closed *iff for any two segment-level events $s_1, s_2 \in \#^h_{sev}(h)$ any event $x \in E^s(\#^h_{src}(h), \#^h_{tgt}(h)), s_1 <^s x <^s s_2$ is also in h, i.e., $x \in \#^h_{sev}(h)$. The order $<^h$ of system-level events is acyclic and must respect time: $<^h \subseteq \{(h_1, h_2) \in E^h \times E^h \mid \#^s_{start}(s_1) < \#^s_{start}(s_2) \vee \#^s_{end}(s_1) < \#^s_{end}(s_2)\}$.*

Figure 2 indicates two closed system-level events h_4 and h_5, i.e., two closed sets of consecutive segment-level events. In a real BHS, the higher load in $(L4, L5)$ may *cause* the conveyor belt $(L3, L4)$ to stop.

In contrast to Def. 2, Def. 3 allows two design decisions. (1) *Which sets of segment-level events shall be aggregated into a system-level event?* (2) *Which system-level events might be causally related and hence should be ordered by $<^h$?* These design decisions correspond to our research questions (1) and (2) which we answer in Sect. IV and V-B, respectively. There we will call sets of system-level events connected by $<^h$ a *cascade* of system-level behavior. We then show in Sect. VI that we can answer research question (3) be searching for frequent patterns in $L^h = (E^h, \#^h, <^h)$ along $<^h$.

IV. Detecting System-level Events

We consider how to detect "outlier" system-level events E^h of interest to build a system-level event log $L^h = (E^h, \#^h, <^h)$ from a normal case-level event log L^c, i.e., research question (1). We already obtained the segment-level event log $L^s = (E^s, \#^s, <^s)$ of L^c as described in Sect. III and shown in Tab. II. We are interested in detecting *closed* system-level events h, i.e., sets of consecutive segment-level events, that together show a "different" behavioral characteristic than other events in the segment, as shown in Fig. 2.

Detecting such differences requires a separate detection method for each type of behavioral characteristic. Thus, solving this problem requires the analyst to define a set of system-level event detection methods $D = \{detect_1, ..., detect_k\}$

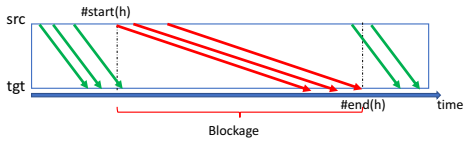


Fig. 3: Segment-level events forming a system-level blockage.

where each detection method $detect_i(E^s, \#^s, <^s) = E^h_i$ returns a set of system-level events of a specific type according to Def. 3. The set of system-level events is then $E^h = \bigcup_{detect_i \in D} detect_i(E^s, \#^s, <^s)$.

In the following, we define two specific system-level event detection methods to detect dynamic bottlenecks (Sect. IV-A) and high-load situations (Sect. IV-B) for the domain of material handling systems (MHS).

A. Detecting Dynamic Bottlenecks as Blockages

A dynamic bottleneck emerges when one or more cases take significantly higher times to flow from activity a to activity b than the baseline duration for segment (a, b). If $\tilde{t}$ is the baseline duration for cases traversing (a, b), then we can observe a *significant delay* as a segment-level event $s \in E^s(a, b)$ where the time-duration $\Delta t(s) = \#^s_{end}(s) - \#^s_{start}(s)$ is significantly higher $\Delta t(s) \gg \tilde{t}$ than the baseline.

In highly optimized systems such as MHS, the median duration in a segment does form the intended baseline, i.e., $\tilde{t}$ is the median of $\Delta t(E^s) = \{\Delta t(s) \mid s \in E^s\}$. Any segment-level event s where $\Delta t(s)$ is an outlier wrt. $\Delta t(E^s)$ in a statistical sense would show a significant delay. We analyzed and compared 4 univariate outlier detection methods and identified the *modified z-score* as the most reliable technique. [22, Ch. 5.2]

The modified z-score relates a concrete deviation $(\Delta t(s) - \tilde{t})$ of a segment-level event s to the median absolute deviation $MAD = \text{median}_{s \in E^s}(|\Delta t(s) - \tilde{t}|)$ of all events:

$$M(s) = (0.6745 \cdot (\Delta t(s) - \tilde{t}))/MAD \quad (1)$$

The factor 0.6745 allows approximating a standard normal distribution. If M_s is higher than a threshold k_{delay}, the $\Delta t(s)$ is an outlier wrt. $\Delta t(E^s)$ and we consider the case $\#^s_{id}(s)$ to suffer a significant delay, i.e., $delay(s) = true$ iff $M_s > k_{delay}$. For the domain of material handling systems, we confirmed normality of the data and identified $k_{delay} = 50$ as reliable together with domain experts. [22, Ch. 5.2]

A *dynamic bottleneck* emerges when several consecutive cases experience a delay. For example, in a MHS it may be that multiple cases are stuck behind each other in the same queue (red cases in Fig. 3) while other cases wait either before the blocked queue or are routed around and only enter the queue after the blockage ends (green cases at the end of Fig. 3). We can formally describe a blockage as a set of consecutive segment-level events s with $delay(s) = true$.

We detect all blockages in a segment (a, b) as follows. Assuming all start time-stamps to be distinct, $E^s(a, b) =$

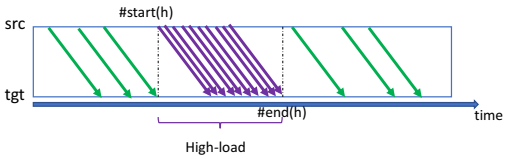


Fig. 4: Segment-level events forming high-load at system-level

$\langle s_1, \ldots, s_n \rangle$ defines a sequence of segment-level events. Iterating over $\langle s_1, \ldots, s_n \rangle$ we find all subsequences $\langle s_v, \ldots, s_w \rangle$ with $delay(s_i) = true, v \leq i \leq w$. Each such subsequence defines a new system-level event h of $\#^h_{type}(h) = blockage$ over $\#^h_{sev}(h) = \{s_v, \ldots, s_w\}$ (see Def. 3).

Suppose that in our running example $(L3, L4)$ has a median duration of $\tilde{t} = 10secs$ and $MAD = 1sec$. Then for $k_{delay} = 50$, segment-level events s_{78}, s_{79}, s_{80} of Tab. II have $delay(s_i) = true$, e.g., $M(s_{78}) = 0.6745 \cdot (125 - 10)/1 = 77.56 > k_{delay}$, resulting in the blockage (BL) system-level event h_5 in Tab. III.

B. Detecting High-load

A process having to handle a significantly higher number of cases in a process step may face performance problems. We can quantify the workload in a segment during a time-interval as the number of segment-level events crossing this time interval. If this workload is significantly above a baseline workload, then this time-interval describes a *high-load* outlier situation as illustrated in Fig. 4.

To detect high-load events in segment (a, b), we bin all segment-level events into bins of length k_{bin}, i.e., $bin_i = \{s \in E^s(a, b) \mid \#^s_{start}(s) \in [i \cdot k_{bin}; (i + 1) \cdot k_{bin})\}$ with $load_i = |bin_i|$. We then use the IQR-method to classify bins as outliers wrt. load. Let $P^{75}_{(a,b)}$ be the 75th percentile of the $load_i, i > 0$. For a bin holds $highload(bin_i) = true$ iff $load_i > P^{75}_{(a,b)}$. As for blockages, we search for subsequences $\langle bin_v, \ldots, bin_w \rangle$ with $highload(bin_i) = true, v \leq i \leq w$. Each such subsequence defines a new system-level event h of $\#^h_{type}(h) = highload$ over $\#^h_{sev}(h) = bin_v \cup \ldots \cup bin_w$ (see Def. 3).

Suppose that in our running example segment $(L4, L5)$ for $k_{bin} = 1min$ has $P^{50} = 2$ and $P^{75} = 3$, i.e., 75% of the bins see at most 3 bags per minute. Then $s_{73}, s_{74}, s_{76}, s_{77}$ of Tab. II fall into a bin with $load = 4$, resulting in the high-load (HL) system-level event h_4 in Tab. III.

V. Detecting System-Level Event Cascades

In Sect. IV, we detected system-level events E^h from a given case-level event log L^c, for example the events in Tab. III. In this section, we detect which system-level events might be causally related, i.e., research question (2). We first discuss how to identify whether two events in E^h are correlated in Sect. V-A. Ordering correlated events by time yields the relation $<^h$ of a system-level event log $L^h = (E^h, \#^h, <^h)$ (see Sect. III). We then observe that a set of events connected by $<^h$ forms a *cascade* of system-level behavior; we formalize this in Sect. V-B.

TABLE III: Detected System-Level Events

	type	*src*	*tgt*	*start*	*end*	*sev*
h_1	HL	L1	L2	10:20	10:23	$s_{65}, \ldots, s_{68}$
h_2	HL	L2	L3	10:21	10:23	$s_{61}, \ldots, s_{64}$
h_3	HL	L3	L4	10:22	10:24	$s_{69}, \ldots, s_{71}$
h_4	HL	L4	L5	10:23	10:24	$s_{73}, s_{74}, s_{76}, s_{77}$
h_5	BL	L3	L4	10:23	10:26	s_{78}, s_{79}, s_{80}
h_6	HL	L5	L6	10:25	10:26	$s_{82}, \ldots, s_{84}$

A. Correlating System-Level Events

In contrast to case-level events, the system-level events E^h lack a unique case identifier that describes which events belong to the same *observable* dynamic. Rather, we infer from emergent system-level event properties whether two events might be connected by a *direct material cause* that explains why one event has an influence on another event. The example in Fig. 2 illustrates this: h_4 and h_5 originate from segment-level events of disjoint sets of cases, yet the high-load h_4 in segment $(L4, L5)$ prevents cases on $(L3, L4)$ from entering $(L4, L5)$ which causes the blockage h_5 in $(L3, L4)$. In the following, we discuss which emergent properties we can use to explain possible causal relations between system-level events.

Each system-level event h in our event model has two emergent properties at the system-level: a *spatial property* (it is located on segment $(\#^h_{src}(h), \#^h_{tgt}(h))$) and a *temporal property* (it happens in the interval $[\#^h_{start}(h); \#^h_{end}(h)]$). There can only be a direct material cause between two system-level events if they are "sufficiently" close to each other in space and time, i.e., events that are too far apart cannot be related. For example, in Fig. 2, h_4 and h_5 are spatially close (share activity $L4$) and temporally close (overlap in time).

We propose to use a spatial distance measure $d_s : E^h \times E^h \to D_s$ and a temporal distance measure $d_t : E^h \times E^h \to D_t$ to characterize the distance between two events by $d_s(h_1, h_2)$ and $d_t(h_1, h_2)$. From domain-knowledge we can derive which spatial and temporal distances together are *Close* $\subseteq D_s \times D_t$. Then two events $h_1, h_2 \in E^h$ are correlated when $(d_s(h_1, h_2), d_t(h_1, h_2)) \in$ *Close*. Note that distance does not have to be continuous.

- Measuring spatial distance: The spatial distance measure d_s could be based on the physical distance of the locations of h_1 and h_2 or whether particular materials or information is being exchanged. For the case of a MHS with physical conveyor belts, we assume that two system-level events are correlated more likely if they happen on neighbouring conveyor belts. We define distance $d_s(h_1, h_2)$ as "proximity" in terms of locations shared between h_1 and h_2, i.e., $d_s(h_1, h_2) = |\{\#^h_{src}(h_1), \#^h_{tgt}(h_1)\} \cap \{\#^h_{src}(h_2), \#^h_{tgt}(h_2)\}|$. We consider h_1 and h_2 spatially close if they are spatially connected, $d_s(h_1, h_2) > 0$.
- Measuring temporal distance: The temporal distance measure d_t could be based on time distance or the amount of overlap between the intervals h_1 and h_2. For the case of a MHS, we consider system-level events as

related if they overlap in time or contain each other in terms of Allen Algebra. Formally, $d_t(h_1, h_2)$ is the Allen relation [23] that holds for $[\#^h_{start}(h_1); \#^h_{end}(h_1)]$ and $[\#^h_{start}(h_2); \#^h_{end}(h_2)]$; specifically $[t_1; t'_1]$ *overlaps* with $[t_2; t'_2]$ iff: $t'_2 - t'_1 > 0$ and $t_2 - t_1 > 0$; $[t_1; t'_1]$ *contains* $[t_2; t'_2]$ iff: $t'_1 - t'_2 > 0$ and $t_2 - t_1 > 0$. We define that h_1 and h_2 are close if $d_t(h_1, h_2) \in \{overlaps, contains\}$. Note that $d_t(h_1, h_2)$ defines that h_1 and h_2 are close only if h_1 starts before h_2, i.e., $d_t(h_1, h_2)$ is directed.

Altogether, two events h_1, h_2 are correlated iff they are spatially and temporally close, i.e., $d_s(h_1, h_2) > 0$ and $d_t(h_1, h_2) \in \{overlaps, contains\}$. In Fig. 2, h_4 and h_5 are correlated.

B. Cascades of Correlated Events

We now want to order the correlated system-level events over time to study system-level behavior. Technically, we will end up defining an acyclic order $<^h$ of a system-level event log (Def. 3). We conceptualize this problem as a directed graph $G = (E^h, F^h)$ where each system-level event is a node $h \in E_h$. There is a directed edge $(h_1, h_2) \in F^h$ between any two events that describes that h_1 and h_2 potentially occur in the same dynamic(directed clique). From this graph, we retain only those edges (h_1, h_2) where h_1 and h_2 are spatially and temporally close as described in Sect. V-A.

In other words, we *query* the graph G for the sub-graph of edges (h_1, h_2) where $(d_s(h_1, h_2), d_t(h_1, h_2)) \in Close$. Each queried edge (h_1, h_2) states that h_1 and h_2 are correlated by overlapping spatially and temporally *and* h_1 starts before h_2. This satisfies our requirement for the acyclic ordering relation $<^h$ of system-level event logs (Def. 3). We thus can define $<^h = \{(h_1, h_2) \in E^h \times E^h \mid (d_s(h_1, h_2), d_t(h_1, h_2)) \in Close\}$. The resulting system-level event log $L^h = (E^h, \#^h, <^h)$ can be understood as directed acyclic graph of correlated events. In general, the system-level event log $L^h = (E^h, \#^h, <^h)$ is not connected when interpreting $<^h$ as directed edges over nodes E^h.

Definition 4 (Cascade)**.** *A* cascade *is a connected component in the system-level event log $L^h = (E^h, \#^h, <^h)$. The events in a cascade are acyclically by local correlations that go forward in time (along the correlation edges $<^h$).*

For instance, the conceptual graph for the system-level events in Tab. III has nodes $h_1, \ldots, h_6$ that are all connected to each other. Querying this graph for correlated events returns the edges (h_1, h_2), (h_2, h_3), (h_3, h_4), (h_4, h_5), (h_4, h_6) resulting in the system-level event log of Fig. 5 for the process in Fig. 1. Together they describe that a high-load h_1 in $(L1, L2)$ propagated forward to h_4 in $(L4, L5)$ from where it further propagated in parallel: backwards as a blockage h_5 in $(L3, L4)$ and forward as high-load h_6 in $(L5, L6)$.

VI. Detecting Frequent Patterns in Cascades

Figure 6 shows some cascades of system-level events detected on real-life data of a BHS. These cascades are complex in structure and not all correlated outlier dynamics may be

Fig. 5: Cascade of the system-level events of Tab. III

systematic, that is, occur frequently or even describe cause-effect relations. In the following, we discuss how to identify frequently occurring patterns in the cascades of a system-level event log $L^h = (E^h, \#^h, <^h)$, i.e., research question (3).

Given the graph-based nature of L^h, we propose to describe these patterns as sub-graphs.

Definition 5 (Cascade pattern)**.** *A* cascade pattern *is a connected labeled graph $P = (N^p, \#^p, F^p)$; each node $n \in N^p$ describes a system-level event by type $\#^p_{type}(n)$, and source $\#^p_{src}(n)$ and target $\#^p_{tgt}(n)$ activity; each directed edge (n_1, n_2) describes that n_1 "causes" n_2 in the sense of Sect. III. A cascade pattern P* occurs *in a system-level event log L^h iff P is an isomorphic sub-graph in L^h wrt. $\#^p_{type}(n), \#^p_{src}(n), \#^p_{tgt}(n)$, i.e., ignoring time-intervals of events in L^h.*

Finding all frequent cascade patterns becomes a *frequent sub-graph mining problem*. Frequent sub-graph mining is a data mining task of finding all sub-graphs that appear in at least k_{min} graphs of a graph database. In our situation, L^h is the graph database, each cascade is a graph in L^h, and the frequent sub-graphs are the cascade patterns we are interested in. Various sub-graph mining techniques are available; in our evaluation we use TKG [24] which returns the top-k sub-graphs wrt. their frequency (support) and has low running times also on large problem instances.

VII. Evaluation

We implemented and released the first version of our technique (https://github.com/processmining-in-logistics/cascades/releases/tag/v1.0) and evaluated it on real-life event data of a BHS of a major European airport to answer the following questions. (1) Are there dynamic bottlenecks and do they occur alone or are they correlated? (2) What is the structure and frequency of cascades that precede dynamic bottlenecks? (3) Are there frequent patterns in the cascades that are describing cause-effect relations of a BHS?

(1) In the BHS, each bag is a case and events are recorded when bags pass sensors on conveyors, the sensor location is recorded as activity name. We applied our implementation on an event log which contains 4,220,897 case-level events of 152,518 bags over 7 days. Table IV provides statistics of detected system-level events per day. We distinguish between blockage and high-load events that were isolated (iso) and correlated in a cascade (co): most of the outlier behaviors are correlated. We also report the average delay due to blockages and the maximum number of bags in a blockage (size): the frequency and duration of blockages varies, but overall hundreds of bags are blocked each day, an average 4 minutes of delay may cause bags checked in late to miss their flight.

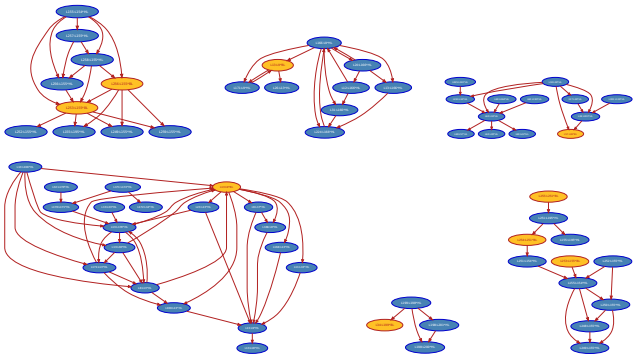


Fig. 6: Detected cascades of system-level events.

TABLE IV: Detected system-level events and cascades

Day	High-load iso	High-load co	Blockage iso	Blockage co	Blockage delay	Blockage size	# Cascades total	# Cascades block.
Mon	1073	15876	23	41	214s	4	586	23
Tue	924	16582	57	115	294s	13	785	40
Wed	920	15719	51	109	243s	9	869	47
Thu	1287	15871	30	71	454s	6	904	38
Fri	1390	18428	55	83	266s	6	1090	50
Sat	1069	16802	38	146	290s	6	1029	55
Sun	1183	16063	59	151	220s	12	991	56

(2) Table IV also reports the number of cascades found in total and cascades with at least one blockage event: most cascades contain no blockage but several cascades contain multiple correlated blockage events. We detect more cascades with and without blockages on Friday to Sunday which is in line with higher flight traffic. Fig. 6 shows a few example cascades: cascades greatly vary in size and complexity.

(3) We applied the frequent subgraph mining algorithm TKG of SMPF [25] on the detected cascades with at least one blockage. We detected 4591 frequent subgraphs; 500 subgraphs contained at least one blockage with a minimum support of 3. We analyzed the structure of the frequent subgraphs and identified the following 4 types, also shown in Fig. 7.

1) Fig. 7a illustrates the first cascade pattern type. Next to each system-level event h_1 we visualize its segment as a node $\#^h_{src}(h_1) : \#^h_{tgt}(h_1)$ connected via a dashed edge. The red edge from one segment node to the next indicates that the target activity of the first is the source of the second activity (they form a sequence). We can see that the causal order $h_1 <^h h_2 <^h h_3$ follows the direction of the segments, i.e., the cascade propagates *forward* in the process.
2) Fig. 7b illustrates the second pattern where $h_1 <^h h_2 <^h h_3$ are ordered in opposite direction of the segments, i.e., the cascade propagates *backwards* in the process.
3) Fig. 7c illustrates the third pattern where the segments are not sequentially ordered but diverge from the same source or converge to the same target, i.e., the cascade propagates *in parallel*.
4) Fig. 7d illustrates the fourth pattern where the cascade propagates *mixed* through the process, i.e., forward, backward, and parallel.

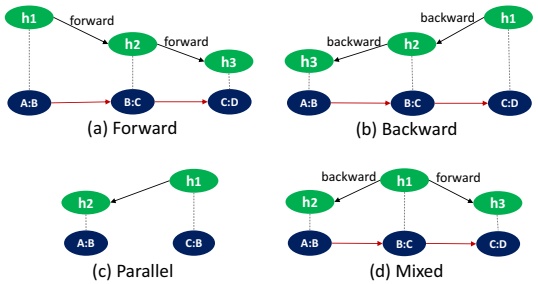


Fig. 7: Frequent types of cascade patterns.

TABLE V: Frequency of detected cascade patterns per type

Patterntype	*Frequency*	*Minsupport*	*Maxsupport*
Forward	50	3	6
Backward	60	3	5
Parallel	68	3	6
Mixed	322	3	4

Table V shows how many patterns of each type were found in the data and their minimum and maximum support. The large number of mixed patterns suggests that dynamic bottlenecks are most often preceded by complex dynamics.

Fig. 8 shows 3 detected *mixed* frequent subgraphs with at least one blockage with the physical layout. Fig 8a shows a simple subgraph detected close to the end of the process. We indicate the average delay between start times of two system-level events on the edge. High-load (h1) on segment $L13$:$L14$ propagates forward (h2) and causes a blockage (h3) after 143 secs at up-stream segment $L16$:$L13$. This pattern is causally explained by the system: $L13$:$L14$ is overloaded causing $L16$:$L13$ to stop forwarding bags. Subgraph 8b was found in baggage screening and is similar to subgraph 8a high-load (h2) on $L3$:$L2$ causes a blockage (h3) at upstream segment $L4$:$L3$. Interestingly (h2) is preceded in this frequent pattern by a parallel high-load (h1). The blockage (h3) is explained causally by (h2) whereas the preceding high-load (h1) is only frequently correlated to (h2) but both causally link to (h4). Fig. 8c illustrates a bigger subgraph detected at the end of the process. Here, high-loads (h1) and (h2) converge via $L6$:$L7$ and $L8$:$L7$ to location $L7$, propagate backwards as high-load (h3) to $L9$:$L8$. Location $L8$ is a split to three different locations $L7$, $L10$, $L11$. The high-load propagates downstream and causes a blockage (h5) on $L8$:$L10$. This complex dynamic occurred repeatedly and the first high-load occurred 194 secs before the blockage.

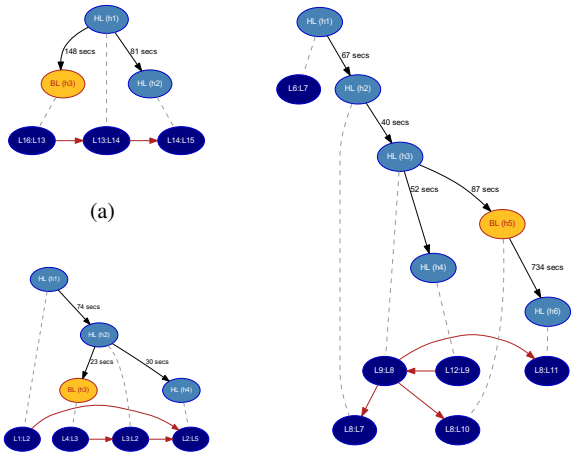


Fig. 8: Examples of detected cascade patterns.

VIII. Conclusion

We presented and validated a new method to detect cascades of system-level behavior which frequently precede dynamic bottlenecks from regular event logs. We could detect dynamic bottlenecks in a highly optimized BHS of a major European airport. The bottlenecks occur in cascades of high-load situations that propagate through the system. We verified that all propagation dynamics are causally explained by the underlying system dynamics.

The current approach has several limitations. The identified patterns occur relatively infrequently in the cascades and show a large variety. All the patterns have low support which requires further investigation. Analysis of longer time scales is required to increase confidence in the findings. Currently, we only consider high-load as an explanation for dynamic bottlenecks. Other causes such as availability of workers and equipment have to be included in future work. We only detect dynamic bottlenecks but would like to predict those bottlenecks, e.g., by splitting patterns in prediction rules with antecedent and consequent. We only demonstrated the applicability of our approach to BHS. The applicability to other systems with strict queuing has to be confirmed while generalization to business processes requires further research.

Acknowledgment

The research leading to these results has received funding from Vanderlande in the project "Process Mining in Logistics". Özge Köroğlu worked at Vanderlande Industries, Veghel, the Netherlands for parts of this study. We thank Marwan Hassani for his valuable input on formulating the research problem in a clear manner.

References

[1] R. W. Hall, "Queueing methods: For services and manufacturing," 1991.

[2] F. Milani and F. M. Maggi, "A comparative evaluation of log-based process performance analysis techniques," in *BIS 2018*, vol. 320 of *LNBIP*, pp. 371–383, Springer, 2018.

[3] T. L. Graafmans, O. Turetken, J. H. Poppelaars, and D. Fahland, "Process mining for six sigma: a guideline and tool support," *BISE*, 2020.

[4] S. Suriadi, C. Ouyang, W. M. P. van der Aalst, and A. H. M. ter Hofstede, "Event interval analysis: Why do processes take time?," *Decis. Support Syst.*, vol. 79, pp. 77–98, 2015.

[5] V. Denisov, D. Fahland, and W. M. P. van der Aalst, "Unbiased, fine-grained description of processes performance from event data," in *BPM 2018*, vol. 11080 of *LNCS*, pp. 139–157, Springer, 2018.

[6] E. L. Klijn and D. Fahland, "Performance mining for batch processing using the performance spectrum," in *BPM 2019 Workshops*, vol. 362 of *LNBIP*, pp. 172–185, Springer, 2019.

[7] F. Folino, G. Greco, A. Guzzo, and L. Pontieri, "Mining usage scenarios in business processes: Outlier-aware discovery and run-time prediction," *Data Knowl. Eng.*, vol. 70, no. 12, pp. 1005–1029, 2011.

[8] X. Lu, D. Fahland, F. J. H. M. van den Biggelaar, and W. M. P. van der Aalst, "Detecting deviating behaviors without models," in *BPM 2015 Workshops*, vol. 256 of *LNBIP*, pp. 126–139, Springer, 2015.

[9] Y. Huang, L. Zhang, and P. Zhang, "A framework for mining sequential patterns from spatio-temporal event data sets," *IEEE Trans. Knowl. Data Eng.*, vol. 20, no. 4, pp. 433–448, 2008.

[10] H. Reguieg, B. Benatallah, H. R. M. Nezhad, and F. Toumani, "Event correlation analytics: Scaling process mining using mapreduce-aware event correlation discovery techniques," *IEEE Trans. Serv. Comput.*, vol. 8, no. 6, pp. 847–860, 2015.

[11] S. Pourmirza, R. M. Dijkman, and P. Grefen, "Correlation miner: Mining business process models and event correlations without case identifiers," *Int. J. Cooperative Inf. Syst.*, vol. 26, no. 2, pp. 1742002:1–1742002:32, 2017.

[12] H. Nguyen, M. Dumas, A. H. M. ter Hofstede, M. L. Rosa, and F. M. Maggi, "Business process performance mining with staged process flows," in *CAiSE 2016*, vol. 9694 of *LNCS*, pp. 167–185, Springer, 2016.

[13] A. Senderovich, J. C. Beck, A. Gal, and M. Weidlich, "Congestion graphs for automated time predictions," in *EAAI 2019*, pp. 4854–4861, AAAI Press, 2019.

[14] S. Nath, H. Yu, P. B. Gibbons, and S. Seshan, "Subtleties in tolerating correlated failures in wide-area storage systems," in *NSDI 2006*, USENIX, 2006.

[15] M. Sedaghat, E. Wadbro, J. Wilkes, S. de Luna, O. Seleznjev, and E. Elmroth, "Diehard: Reliable scheduling to survive correlated failures in cloud data centers," in *CCGrid 2016*, pp. 52–59, IEEE Computer Society, 2016.

[16] D. Tang and R. K. Iyer, "Analysis and modeling of correlated failures in multicomputer systems," *IEEE Trans. Computers*, vol. 41, no. 5, pp. 567–577, 1992.

[17] A. Perer and F. Wang, "Frequence: interactive mining and visualization of temporal frequent event sequences," in *IUI 2014*, pp. 153–162, ACM, 2014.

[18] W. Liu, Y. Zheng, S. Chawla, J. Yuan, and X. Xie, "Discovering spatio-temporal causal interactions in traffic data streams," in *SIGKDD 2011*, pp. 1010–1018, ACM, 2011.

[19] Y. Wu and H. Tan, "Short-term traffic flow forecasting with spatial-temporal correlation in a hybrid deep learning framework," *CoRR*, vol. abs/1612.01022, 2016.

[20] S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan, "Attention based spatial-temporal graph convolutional networks for traffic flow forecasting," in *AAAI 2019*, pp. 922–929, AAAI Press, 2019.

[21] W. M. P. van der Aalst, D. T. G. Unterberg, V. Denisov, and D. Fahland, "Visualizing token flows using interactive performance spectra," in *PETRI NETS 2020*, vol. 12152 of *LNCS*, pp. 369–380, Springer, 2020.

[22] Özge Köroğlu, "Outlier Detection in Event Logs of Material Handling System," Master's thesis, Eindhoven University of Technology, 2019.

[23] J. F. Allen, "Maintaining knowledge about temporal intervals," *Commun. ACM*, vol. 26, no. 11, pp. 832–843, 1983.

[24] P. Fournier-Viger, C. Cheng, J. C. Lin, U. Yun, and R. U. Kiran, "TKG: efficient mining of top-k frequent subgraphs," in *BDA 2019*, vol. 11932 of *Lecture Notes in Computer Science*, pp. 209–226, Springer, 2019.

[25] P. Fournier-Viger, J. C. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng, and H. T. Lam, "The SPMF open-source data mining library version 2," in *PKDD 2016*, vol. 9853 of *LNCS*, pp. 36–40, Springer, 2016.

Identifying and Reducing Errors in Remaining Time Prediction due to Inter-Case Dynamics

Eva L. Klijn
Eindhoven University of Technology
The Netherlands
e.l.klijn@tue.nl

Dirk Fahland
Eindhoven University of Technology
The Netherlands
d.fahland@tue.nl

***Abstract*—Remaining time prediction (RTP) is the problem of predicting the time until a specific process step is reached in a specific process instance. Feature engineering in established RTP techniques assume that cases progress in isolation. Inter-case dynamics such as batching violate this assumption, leading to high prediction errors. Yet, existing RTP techniques do not consider the nature of prediction errors to improve quality. We contribute a technique for identifying the location and context of prediction errors by visually comparing prediction and ground truth. For the case of batching, we show how to engineer inter-case features that detail the impact of batching on the remaining time. Our evaluation shows that adding inter-case features improves prediction performance across almost all evaluated primary prediction methods on two real-life event logs, with error reductions of up to 37%. We finally advocate for a more thorough and transparent evaluation of prediction errors in RTP research, including our own results.**

***Index Terms*—predictive process monitoring, remaining time prediction, inter-case features, batch processing**

I. Introduction

Remaining time prediction (RTP) in processes is the problem of predicting the time until a specific process step is reached in a specific instance of a process. When put to practice successfully, RTP allows for the timely identification of performance problems such as delays in a specific process or workload peaks [1]. RTP is usually accomplished by learning from historic event data a predictive model RM. Given a prefix of the current process instance RM returns the estimated remaining time until completion of the instance.

A survey and benchmark of various RTP techniques developed in the recent years [2] identifies progress and best practices in training RTP models. The survey and the primary studies evaluate model quality using the *mean absolute error* (MAE) between predicted and actual remaining time. Most techniques still suffer from considerable errors: for 13 of the 16 evaluated data sets, the best technique shows average relative errors of 31%-55% and absolute errors of up to 180 days. None of the existing studies, c.f. [2] detail the nature of the errors made nor consider the question of how to systematically overcome the errors and the limitations of the model learning techniques. We consider this lack of evaluation and reflection a gap in RTP research and practice.

We address the problem of *identifying the nature of errors made by a given RTP model RM, and of using these insights to engineer additional features that reduce the errors of RM (after re-training)*. Answers to this question help establishing an effective *feedback loop* that is necessary for improving RTP models in a systematic way.

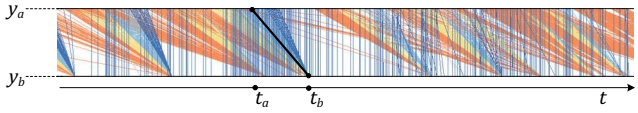


Fig. 1. Performance spectrum example of single process segment.

In this paper, we focus on prediction errors made due to an invalid assumption made by most RTP techniques. All surveyed RTP techniques [2] predict the remaining time using only *intra-case features* of events in the current process instance, thereby assuming each process instance is performed in isolation and its progress is independent of other process instances. The visual analytics technique of the *Performance Spectrum* (PS) [3] refutes this assumption. The patterns visible in the PS in Fig. 1 reveal *inter-case dynamics*, such as *batching* [4], influencing the progress of multiple cases.

To identify which mechanisms and dynamics contribute to high prediction errors of a given model RM, we propose to adapt the PS in two ways. First, we color-code segments in the PS based on whether the prediction error made by RM increases or decreases at a specific process step, revealing at which step RM makes the highest error. Second, we overlay the actual behavior of all cases with the predictions made by RM thereby revealing which behavioral patterns of inter-case dynamics RM does not identify. For the most prevalent pattern of batching, we then show how to engineer so-called *inter-case features* [5]–[7] which allow to inject information about the progress of other cases as context for the current case. We specifically engineer features that describe the waiting time of cases until batching occurs thereby giving more precise information about the remaining time.

In our evaluation, we show that identifying and engineering inter-case batching features using the proposed techniques allows reducing prediction errors by 13%-37% for batched cases without negatively impacting non-batched cases on two real-life event logs. We also critically evaluate our improved prediction models using the proposed techniques and several standard, detailed data plots, thereby outlining shortcomings and limitations of the model and how to improve the models

978-1-7281-9832-3/20/$31.00 ©2020 IEEE
DOI 10.1109/ICPM49681.2020.00015

further using these insights.

Next, we discuss background and related work of RTP in Sect. II. We present our techniques for error identification in Sect. III and show how to engineer inter-case features for batching in Sect. IV. We evaluate models trained on these features and their errors in Sect. V and conclude in Sect. VI.

II. Background & Related Work

A. Remaining Time Prediction Background

In remaining time prediction (RTP) - and PPM in general - predictions are made using the history of ongoing process instances based on the continuations of previously completed process instances. A complete process instance pid is recorded as the sequence of events $\sigma = \langle e_1, ..., e_m \rangle$ that occurred in pid, i.e., the current *trace* of pid. Each event records multiple attributes. We write $\#_n(e)$ for the value of attribute n of event e and require that $\#_{activity}(e)$ and $\#_{time}(e)$ are defined for all events.

For process prediction, an incomplete trace or *prefix* and the information carried by this prefix (e.g. case and event attributes) are used as input to a prediction model that aims to reason about a certain outcome y at a moment in the future. We call such a reasoning step about a specific prefix $hd^k(\sigma) = \langle e_1, ..., e_k \rangle, k \leq |\sigma|$ an *individual prediction* $P_{hd^k(\sigma)}$ or just P if the prefix is clear. We define $a_k = \#_{act}(e_k)$ as the point of prediction and $t_{a_k} = t_P = \#_{time}(e_k)$ as the time moment of prediction at which the predictive reasoning step took place. Remaining time prediction reasons about the time y_P until the last activity $a_{|\sigma|}$ of the trace will occur. Each prediction has a predicted outcome $\overline{y}_P$ and an actual outcome y_P. y_P is unknown when the prediction is made at run-time, but can be derived from the suffix $tl^{m-k}(\sigma)$ of an already completed trace σ for model training.

B. Developing a Remaining Time Prediction Model

Research in RTP developed various approaches to train a model RM that can predict remaining time $\overline{y}_P$ from $hd^k(\sigma)$ (and other input features). *Process aware approaches* use an explicit representation of a process model in the prediction, such as transition systems [8], stochastic Petri nets [9] or queuing models [10]. *Process-agnostic approaches* do not use an explicit representation of a process model, such as machine learning algorithms (ML) [6], [11], neural networks [12] or statistical models [13]. *Hybrid approaches* combine both so that decision points in a process model representation are enriched with process-agnostic models (such as statistical models or other) [14]. While differing in the technical machinery, all approaches largely follow the same method for model development characterized by 3 parameters: *E*ncoding, *B*ucketing, learning *A*lgorithm.

Data preparation. Prior to model learning, the input data, i.e. the set of prefixes, undergoes standard data preparation, e.g. cleaning, feature engineering, feature encoding, and, in the case of process-agnostic and hybrid approaches, *prefix encoding* E (see [2]). A prefix encoding function E transforms each prefix $\langle e_1, \ldots, e_k \rangle$ into a feature vector $\langle X_1, \ldots, X_n \rangle$ such that it is interpretable for a model learning technique (e.g., an ML-algorithm or a neural network). Three basic prefix encodings exist: the data payload of the last event in the trace (last state encoding $E = l$), the aggregated data payload of all events (aggregate encoding $E = a$) or the concatenated data payload of all events (index-based encoding $E = i$). Optionally, *prefix bucketing* B can be applied to divide encoded prefixes into buckets of prefixes with similar properties to train separate prediction models (see [2]), e.g. buckets of prefixes with the same length (prefix-length bucketing $B = p$), buckets with prefixes grouped by a clustering algorithm (cluster bucketing $B = c$), or all prefixes end up in a single bucket $B = s$.

Model learning. Next, a model is built, validated and tuned. For process aware approaches this often encompasses the mining of a process model and evaluating several metrics such as fitness and precision [15] to find the optimal model configuration. For process-agnostic approaches, the building of a model is often known as training; features in the data are used as input to an algorithm that tries to learn an ML-model that generalizes on the input data to fit a certain label [16]. In our experiments, we will use a combination of bucketing- and encoding techniques and two ML-algorithms A, i.e. random forest ($A = r$) and xgboost ($A = x$) to learn ensembles of regression trees. Using parameters (B, E, A), we can characterize an RTP model for instance by $RM = (p, a, x)$.

Evaluation. Finally, the trained model is evaluated regarding its accuracy and tuned using grid search or cross validation to find the optimal model configuration. In practice, the above steps are repeated until a model with sufficient accuracy is found. In research, improvement of the proposed technique over the state-of-the-art in accuracy is evaluated. For this, the state of the art approach serves as a baseline and will, together with the novel method, be run on a set of unseen data to measure the performance, most often in terms of accuracy. Typical accuracy measures used in RTP are the mean absolute error (MAE), the root mean squared error (RMSE) or the mean absolute percentage error (MAPE). All these measures entail first calculating the error for each prediction separately, after which these are aggregated into a single value by calculating some form of the mean.

As we observed in Sect. I, aggregate measures such as MAE, RMSE, cannot reveal the causes for low model accuracy. Therefore in our evaluation, we will not only use the MAE, but also propose additional, more thorough evaluation methods that allow us to explain the prediction error in a *qualitative manner*. Instead of inference from data preparation or modeling choices, we use *direct observation* of how predictions compare to actual outcomes.

C. Incorporating Inter-Case Features

In this paper, we specifically focus on the inclusion of an inter-case perspective in process prediction. Currently, only a few approaches have been introduced that take this perspective into account. The interplay among running cases is first considered for predicting risks in [17], however this method

does not accommodate for remaining time prediction. In [10], queuing theory is used to annotate a transition system with features such as queue-length to predict the remaining time, acknowledging the influence of other queued cases on a case's remaining cycle time.

The problem of incorporating inter-case features in process-agnostic remaining time prediction has first been explored in [5] and extended in [6]. In these works, two approaches approximate workload by deriving the number of open cases that are similar to the one to be predicted at the moment of prediction. The number of similar cases can be derived using domain knowledge or automatically using so-called case proximity metrics. This number is added as additional feature X_{n+1} to the feature vector. In [7], longer waiting times between processing steps are estimated by mining a Markov-state representation (MSR) for the next process step $a_{k+1} = \#_{act}(e_{k+1})$ at prediction time t_P; the MSR estimates the number of other cases waiting to reach a_{k+1} and the times since latest start and completion of a_{k+1}. From the MSR, additional features $X_{n+1}, \ldots, X_{n+l}$ are derived and added to the feature vector.

Similar to [6] and [7], we also explore the incorporation of inter-case features in remaining time prediction using ML-based methods. In comparison, [6] and [7] derive inter-case features using concepts of queuing theory while assuming ideal queuing behavior in the data to be present such as cases being processed individually. We show in Sect. III that first visually identifying causes of low prediction accuracy allows to avoid false assumptions prior to feature engineering, specifically we identify the impact of non-individual processing in batches. We also adopt the principles of [6], [7] to engineer inter-case features, but we draw on data-summarization of batching behavior for this in Sect. IV.

III. Identifying Problematic Inter-Case Dynamics

In this section, we present a means to analyze the output of any RTP approach from an inter-case perspective for uncovering inter-case related causes of high prediction errors. We first introduce two visualization methods based on the performance spectrum in III-A, after which we outline the procedure of using these methods for the analysis in III-B.

A. The Performance Spectrum

We first shortly recall the performance spectrum (PS) [3], since it serves as a building block for the visualization methods presented in this section. The PS visualizes process behavior over time as illustrated in Fig. 1. It describes how cases transition from activity a to activity b over time t. Whenever a is directly followed by b in a case, we observe an *occurrence* of the *segment* (a, b) from time t_a (moment of occurrence of a) to time t_b. Each occurrence (a, b) is plotted as a line from (t_a, y_a) to (t_b, y_b); revealing patterns in the data not observable under aggregation – specifically any *inter-case dynamics*.

1) Overlaid Performance Spectrum (OPS): We now want to qualitatively evaluate a set of predictions $\mathcal{P}$ by a trained RTP model RM on a test data set by comparing to what extent

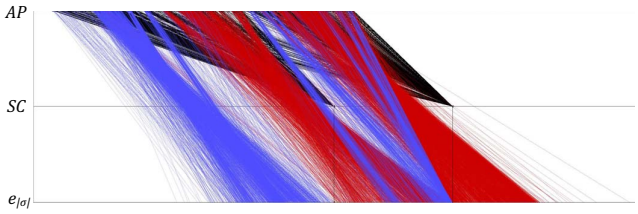


Fig. 2. OPS diagram of RF log for $S = (AP, SC)$ and $k = 4$.

RM can reproduce temporal patterns present in the test data. We propose the *Overlaid* PS that visualizes all remaining time predictions $P \in \mathcal{P}$ overlaid on the actual PS as in Fig. 2.

For overlaying, we focus on a subset of predictions $\mathcal{P}_{k,a,b} \subseteq \mathcal{P}$ at prefix-length k, last activity $a_k = a$ (prediction point), and next activity $a_{k+1} = b$. For each prediction $P \in \mathcal{P}_{k,a,b}$, we visualize (1) its actual progression from activity a to b in segment (a, b), (2) its actual progression from activity b to case completion at $a_{|\sigma|}$ in a "virtual" segment $(b, a_{|\sigma|})$, overlaid with (3) its predicted progression from a to completion $a_{|\sigma|}$ in "virtual" segment $(a, a_{|\sigma|})$. For each prediction $P \in P_{k,a,b}$ we plot occurrences in (a, b) and $(b, a_{|\sigma|})$ using time-coordinates $t_a = t_{a_k} = t_P$, $t_b = t_{a_{k+1}}$, and $t_{a_{|\sigma|}} = t_a + y_P$ from the test data. We then overlay the predictions as lines in $(a, a_{|\sigma|})$ using time-coordinates t_a and $\overline{t}_{a_{|\sigma|}} = t_a + \overline{y}_P$ (predicted absolute end time); see [18] for details.

Fig. 2 shows an OPS diagram of the RF log for $k = 4$ for segment *Add Penalty:Send for Credit Collection* $(a, b) = (AP, SC)$ based on the prediction results of an RTP model $RM = (p, a, x)$ [2] discussed in Sect. II. In Fig. 2, the segment occurrences (AP, SC) and $(SC, a_{|\sigma|})$ in black describe the actual progression of predictions $P \in \mathcal{P}_{4,AP,SC}$ and the *overlaid* segment occurrences $(AP, a_{|\sigma|})$ in red and blue describe their corresponding predicted progression (red for over-prediction and blue for under-prediction). We directly see that RM cannot predict the batching dynamics causing a large prediction error.

2) Performance Spectrum with Error Progression: The OPS globally visualizes the prediction error but not at which prediction points these errors are made. We therefore introduce the PS *with error progression* (PSw/EP). We color code each segment occurrence in the original PS based on whether the prediction error increased or decreased when passing through the segment.

We fix a trace variant $a_1, \ldots, a_s$ along which we analyze error progression. For each segment $(a_k, a_{k+1}), 1 \leq k < s$ and for each trace σ in the test set passing through (a_k, a_{k+1}), we calculate the relative absolute error rae_k, rae_{k+1} of the predictions $P_k = P_{hd^k}(\sigma)$ and $P_{hd^{k+1}}(\sigma)$. We use relative error $rae_k = |\overline{y}_{P_k} - y_{P_k}|/y_{P_k}$ to make errors in different segments comparable. When plotting an occurrence in a segment $(a_k, a_{k+1}), 1 \leq k < s$, we color the line red if the error decreases, i.e., $rae_k > rae_{k+1}$, and blue if the error increases, i.e., $rae_k < rae_{k+1}$.

Fig. 3 shows a PSw/EP diagram of five process segments

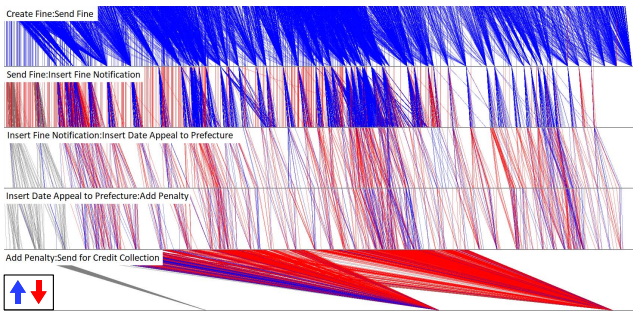


Fig. 3. PSw/EP diagram of RF log for segments (CF, SF), (SF, IF), (IF, ID), (ID, AP) and (AP, SC).

from the RF log. We observe that the majority of the cases in the first segment (CF, SF) is colored blue (taking this step increases the error), while in the last segment (AP, SC) is colored red (taking this step reduces the error).

B. Problematic Pattern Identification

We explain how OPS and PSw/EP allow us to identify regular inter-case dynamics not captured by a prediction model; these regular inter-case dynamics can then be used in subsequent feature engineering.

Generally, we focus on segments with a significant number of occurrences in the test data to ensure sound conclusions from sufficient data. The first step is to see whether there are inter-case dynamics in the test data that impact the actual outcome but are not picked up by the prediction model. Visualizing each significant segment in a separate OPS allows to identify such a mismatch. For example, the OPS in Fig. 2 reveals that RM does not capture batching dynamics.

Next, we use the PSw/EP to locate where in the process most prediction errors are made: a segment (a_k, a_{k+1}) with many red-colored occurrences indicates that a high prediction error at a_k reduces when transitioning to a_{k+1}. Thus, the segment (a_k, a_{k+1}) contains important information about the remaining time that is not available to the prediction model RM, e.g., segment $S = (AP, SC)$ in Fig. 3. We see a specific pattern of lines starting at various points at AP and converging to a single point at SC; the pattern taxonomy of Denisov et al. [3] identifies this emergent inter-case behavior as *end-batching*. Having identified a segment $S = (AP, SC)$ as prediction point with high error and a pattern $R = batching$ as behavioral feature missing in RM, we can now turn to engineer features for R in S to reduce the error.

IV. Creating Inter-Case Features for Batching

In this section, we provide a methodology for creating inter-case features for batching using the output of the previous section, namely a segment S that is subject to high prediction errors caused by batching R. We show how to leverage insights into S and R to engineer a new inter-case feature for remaining time prediction. The feature will estimate for a case that reached the top-part of segment S how long the case will be in S - based on its position relative to all prior cases in S and the "shape" of the batches R. In Sect. V we show that adding this feature to the data improves precision of a remaining time prediction model RM.

TABLE I
Intra-case features with added (meta-) inter-case features.

Activity X_1	Time X_2	... ...	... X_n	c_S X_{n+1}	c_R X_{n+2}	d X_{n+3}	y	$\overline{c}_S$ $\overline{X}_{n+1}$	$\overline{c}_R$ $\overline{X}_{n+2}$	$\overline{d}$ $\overline{X}_{n+3}$	$\overline{y}$
A	21-5 13:17	...	...	0	0	-	127	0	0	-	130
C	24-5 14:05	...	...	1	1	254	270	1	1	210	256
C	30-5 13:08	...	...	1	1	187	178	1	0	-	206
C	15-7 12:36	...	...	1	0	-	287	0	0	-	301
B	21-7 16:13	...	...	0	0	-	301	0	0	-	247
C	27-7 18:04	...	...	1	1	203	183	1	1	186	191
C	05-8 12:20	...	...	0	0	-	93	1	1	142	80
A	06-8 21:00	...	...	0	0	-	45	0	0	-	44

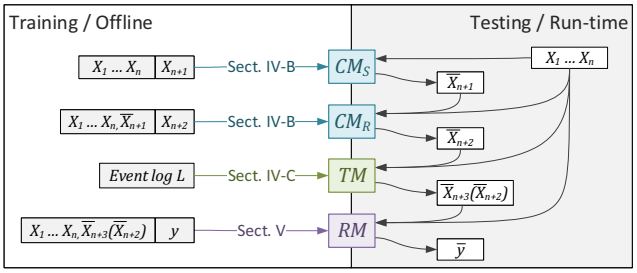


Fig. 4. Overview of feature creation steps.

A. Overview

Table I shows an example table of features used to train a model RM. The columns $X_1, ..., X_n$ show the existing intra-case features. We leverage S and R by consecutively adding columns to the existing features that indicate for each prefix (X_{n+1}) whether it will go through segment S, (X_{n+2}) whether it will also go through batching pattern R, and (X_{n+3}) how long it will last in S as a result of the pattern R. Conceptually, we can then use these three features as additional features to learn a new, better prediction model for the remaining time y.

Features c_S (X_{n+1}), c_R (X_{n+2}), d (X_{n+3}) lie in the future of any prediction P and thus require their own prediction models CM_S, CM_R and TM, respectively. Fig. 4 gives an overview of the different steps necessary to build these models (offline) and the steps that need to be executed at run time.

In the offline phase, we use historic data (middle columns in Table I) to:

1) Learn a classification model CM_S that predicts c_S (Sect. IV-B).
2) Learn a classification model CM_R that predicts c_R (Sect. IV-B).
3) Create a model TM that predicts d (Sect. IV-C).
4) Learn a regression model RM that predicts y.

At run-time, we:

1) Use CM_S to predict $\overline{c}_S$ from $X_1...X_n$.
2) Use CM_R to predict $\overline{c}_R$ from $X_1...X_n, \overline{c}_S$.
3) Use TM to predict $\overline{d}$ from $X_2, \overline{c}_R$.
4) Use RM to predict the remaining time $\overline{y}$ from $X_1...X_n, \overline{d}(\overline{c}_R)$. Here we specifically choose to only use

the feature $\overline{d}$ in the prediction if $\overline{c}_R = 1$. Prefixes for which $\overline{c}_R = 0$ will not get a feature $\overline{d}$.

Clearly, at run-time only the predicted features $\overline{c}_S$, $\overline{c}_R$, $\overline{d}$ are available when predicting $\overline{y}$, thus RM has to be evaluated using predicted values as input. However, our experiments have shown that we obtain better performance also when *training* on the predicted input features $\overline{c}_S$, $\overline{c}_R$, $\overline{d}$.

B. Next Segment Prediction and Batch Prediction

By the analysis in Sect. III, we identified a segment $S = (a, b)$ for which we want to reduce the prediction error.

Let $\mathcal{P}_a$ be all prefixes that reached prediction point a. As not all cases in $\mathcal{P}_a$ will also take segment (a, b), we have to predict the subset $\mathcal{P}_{(a,b)} \subseteq \mathcal{P}_a$ of prefixes which will have b as their next activity. Essentially, this is an instance of the next-activity prediction problem and we can use an existing method [12] to train a model CM_S to predict $\overline{c}_S = 1$ ($P \in \mathcal{P}_{(a,b)}$) or $\overline{c}_S = 0$ as illustrated in Table I.

To predict which of the prefixes in $\mathcal{P}_{(a,b)}$ will also be batched we learn the classification model CM_R. For training, we use the batch miner from [4] to detect the set of cases $\mathcal{P}_{(a,b),R} \subseteq \mathcal{P}_{(a,b)}$ that are part of a batch in segment $S = (a, b)$. We then can learn a classification model CM_R to predict from $X_1, ..., X_n, \overline{c}_S$ whether a prefix P is batched or not, i.e., $\overline{c}_R = 1$ when $P \in \mathcal{P}_{(a,b),R}$, as illustrated in Table I.

C. Distance-to-Batch Prediction

We now discuss how to build a model TM that predicts for each prefix $P \in \mathcal{P}_{(a,b),R}$ that reached a and will be batched in $S = (a, b)$ how long P remains in S until it is processed in a batch R at activity b. Theoretically, this is the remaining time between arrival in S (at time t_a) and departure (at time t_b). Predicting $d = t_b - t_a$ would significantly increase accuracy of the prediction of the remaining time (see Sect. V). However, as t_b is unknown at run-time and batch characteristics differ (see Sect. III), we have to fall back to predicting an approximate "distance" until the next batching moment.

For each prefix, TM has to identify the specific batch the prefix will be part of (we call this the *batch context*), and the distance until that batch closes. In Sect. IV-C1, we use context parameters of all batches identified in S to estimate whether a prefix will be included in the currently open batch (or in a later one). In Sect. IV-C1, we then derive for each prefix the distance until the closing of "its" batch.

1) Deriving Batch Context Parameters: To derive the batch context of a prefix in $\mathcal{P}_{(a,b),R}$ (i.e., in which concrete batch it will end up), we need parameters that describe the batching behavior. For this we will use some of the batching parameters that can be discovered with the existing batch mining technique [4]. Fig. 5 illustrates these parameters for the i-th batch b_i in a segment. Time $t_{i,start}$ and $t_{i,end}$ at the start activity a of the segment are the arrival times of the first and of the last case in batch b_i, respectively. Time BM_i is the batching moment where all cases of batch b_i are processed together at step b; the batch interval BI_i measures the distance to the previous batching moment, i.e. $BI_i = BM_i - BM_{i-1}$; $W_{i,min}$

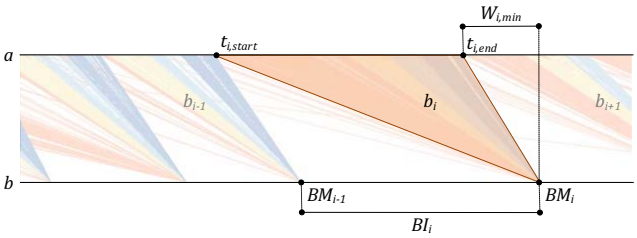


Fig. 5. Batch in performance spectrum annotated with batching parameters.

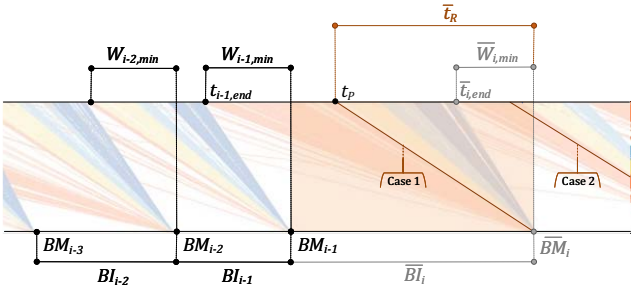


Fig. 6. Illustration of an arrival period annotated with past parameter values (black), forecast parameter values (grey) and forecast time in S subject to R for case 1 (orange).

measures the duration between the last case entering the batch at a and the batching moment. These parameters together describe the triangular shape of one batch b_i with vertices $(a, t_{i,start})$, $(a, t_{i,end})$ - where $t_{i,end} = BM_i - W_{i,min}$ - and (b, BM_i), as well as its relative position to a preceding batch b_{i-1}. We only describe non-overlapping, adjacent batches here, i.e., $t_{i,start} = t_{i-1,end}$, which will omit only a minority of cases from a batch as can be seen in Fig. 5.

We use the batch miner [4] to identify from the performance spectrum data [19] of segment S all batches $b_1, ..., b_n$ and the above parameters. To derive the batching context for a case arriving at a at run-time, we essentially need to predict:

1) When the current batch b_i will close, i.e. predict $\overline{BI}_i$ so we can derive the predicted next batch moment $\overline{BM}_i = BM_{i-1} + \overline{BI}_i$ using the last known batching moment BM_{i-1}.
2) Whether it will be included in the current batch b_i, i.e. $t_P = t_a < t_{i,end}$, or a later batch, i.e. $t_P > t_{i,end}$, i.e. predict $\overline{W}_{i,min}$ so we can derive $\overline{t}_{i,end} = \overline{BM}_i - \overline{W}_{i,min}$.

Note that at run-time the parameters of a batch b_i are only known *after* the batch closed. When making a prediction P for a case reaching the start of segment a at time t_P we therefore have to infer the most likely parameters $\overline{BI}_i$, $\overline{W}_{i,min}$, for the current batch b_i from the parameters of the previous batches $b_{i-1}, b_{i-}, ...$ as illustrated in Fig. 6.

We derive parameters $\overline{BI}_i, \overline{W}_{i,min}$ of the current batch b_i (annotated in grey in Fig. 6) from the parameters of the preceding batches $b_1, ..., b_{i-1}$ through *forecasting*. Forecasting encompasses a collection of techniques that aim to construct a time series model that estimates a trend in the data [20].

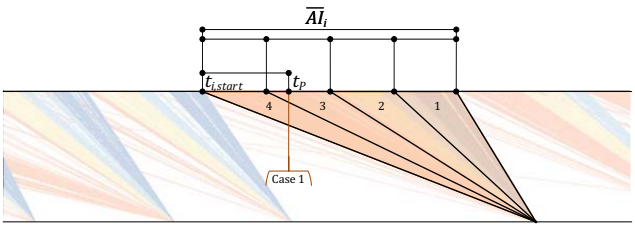


Fig. 7. Illustration of a batch partitioned into $n = 4$ equal parts.

We specifically use *exponential smoothing*, which takes the observations of past periods and applies exponentially decreasing weights over time to calculate a forecast for the next period using a smoothing factor α. We use exponential smoothing with hyperparameter α to train different models F_{BI} and $F_{W_{min}}$ that can forecast $\overline{BI}_i$ and $\overline{W}_{i,min}$ based on past observations $BI_{i-h}...BI_{i-1}$ and $W_{i-h,min}...W_{i-1,min}$, respectively. We then use $\overline{BI}_i$ and $\overline{W}_{i,min}$, together with the previous batch moment BM_{i-1}, to derive the forecast point in time until which cases will still be included in the current batch: $\overline{t}_{i,end} = BM_{i-1} + \overline{BI}_i - \overline{W}_{i,min}$.

2) Deriving Features: We use the above observed and forecast batch parameters (shown in black and grey in Fig. 6, respectively) to create a feature $\overline{d}$ describing the "distance" of any current case $t_P < \overline{t}_{i,end}$ to the expected batching moment $\overline{BM}_i$ (such as cases 1 and 2 in Fig. 6). We discuss two formats for d: time units until batching or "batch partitions", which are both explained in the following and perform differently on different logs, see Sect. V.

a) Time Until Batching: To calculate the time until batching $\overline{d} = \overline{t}_R$ for a case, we consider whether the case arrived before or after the (predicted) closing time $\overline{t}_{i,end}$ of the current batch (see Fig. 6). If a case arrived before closing of the current batch b_i, i.e. $t_P \leq \overline{t}_{i,end}$, it ends up in the current batch (case 1 in Fig. 6) and we predict $\overline{t}_R = BM_{i-1} + \overline{BI}_i - t_P$. Otherwise, i.e. $t_P > \overline{t}_{i,end}$, the case ends up in the next batch (case 2 in Fig. 6) and we predict $\overline{t}_R = BM_{i-1} + 2 \cdot \overline{BI}_i - t_P$ assuming the batching interval to the next batch to remain the same. In the exceptional case where the case is also not predicted to be in the next batch b_{i+1}, i.e. if it arrived after $\overline{t}_{i+1,end} = BM_{i-1} + 2 \cdot \overline{BI}_i - \overline{W}_{i,min}$, then $\overline{t}_R = BM_{i-1} + 3 \cdot \overline{BI}_i - t_P$.

b) Batch Partition: As an alternative to a concrete value for $\overline{d}$, we also propose an abstraction thereof for which we will use the batch partition $\overline{d} = \overline{p}_n$ (when dividing $\overline{BI}_i$ into n parts). Fig. 7 shows how this abstraction is derived.

We divide the arrival interval $[t_{i,end}; t_{i,start}]$ of a batch b_i into n equal parts and then derive in which of these parts a case arrived based on the arrival time and prediction moment t_P of that case within S. Because we assume $t_{i,start} = t_{i-1,end}$, we can predict the length of the arrival interval $\overline{AI}_i = \overline{t}_{i,end} - t_{i-1,end}$ after deriving $\overline{t}_{i,end}$ as explained in Sect. IV-C1.

We divide $\overline{AI}_i$ into n equal parts and define the feature $\overline{p}_n$ for a case arriving at a at time t_P as the number of the partition of $\overline{AI}_i$ it arrives in. For this, we derive the *relative* position of t_P in the interval $\overline{AI}_i$ as $f_P = (t_P - t_{i,start})/\overline{AI}_i$, see Fig. 7. If $j \cdot \overline{AI}_i/n \leq f_p < (j+1) \cdot \overline{AI}_i/n$ then the case arrived in partition $n-j, j = 0, ...n-1$. We define the feature $\overline{p}_n$ as the batch partition in which the case arrives, where n is the number of partitions. These partition numbers are decreasing towards BM_i, to be in line with the regression problem of predicting shorter remaining times if the case is closer to BM_i.

Figure 7 shows how the arrival period $\overline{AI}_i$ is partitioned into $n = 4$ equal parts and additionally highlights the arrival of a case. In this example, we would create a feature with value 3 for case 1.

V. Evaluation

We conducted several experiments to investigate the performance of our method on two real-life event logs [21], [22].

A. Implementation and Computation Time

We implemented the different steps for creating features of Sect. IV in Python 3.7. The scripts, available at https://github.com/multi-dimensional-process-mining/psm-batch-feature-engineering, take an event log, corresponding batch parameters [4] and the "is batched" prediction $\overline{c}_R$ as input and return a labeled event log with additional inter-case feature $I \in \{\overline{t}_R(\overline{c}_R), \overline{p}_n(\overline{c}_R)\}$ for predicting the remaining time $\overline{y}$. We measured an average computation time of 4ms per feature for the BPIC'20 log [22] and 13ms per feature for the RF log [21].

B. Experimental Setup and Procedure

Our objectives are to: (1) compare the performance of different inter-case features $I = \overline{t}_R(\overline{c}_R)$ and $I = \overline{p}_n(\overline{c}_R), 1 < n \leq 20$ against a baseline $I = \varnothing$ and against a "ceiling" $I = t_R(c_R)$ which uses the actual time-to-batch as additional feature. (2) To qualitatively review the prediction models in line with Sect. III. (3) To evaluate the robustness of the best-performing inter-case features across different prediction methods (B, E, A) with $B \in \{c, p, s\}$, $E \in \{a, i, l\}$ and $A \in \{x, r\}$ (see Sect. II) using the implementation from [2].

We selected the RF log [21] and BPIC'20 log [22] after inspecting prediction results of existing baseline technique [2] with the techniques in Sect. III showcased batching features appropriate for our method of Sect. IV. In the RF log, several segments show batching. We selected all segments ending in *Send for Credit Collection*, $(* : SC)$, as these contain only end-batches of > 5300 cases on average, processed ~ 1 year apart. In the BPIC'20 log, two segments show batching. We picked the segment *Request Payment:Payment Handled* (RP, PH), which occurs latest in the process and contains end-batches of highly deviating size (avg. 48, stdev. 38) strictly processed on Tuesdays and Thursdays. For each labeled event log, the first 80% and last 20% (based on a temporal split) are then used for the training and testing, respectively. For the RF log, we used the implementation from [23] to build classification model $CM_{S=(*:SC)}$ (with an accuracy of 72%) and since all cases are batched in the segment, we did not require a model CM_R. For the BPIC'20 log, we required

TABLE II
MAE OF INTER-CASE FEATURES I FOR RF LOG ($RM = s, l, r$) AND BPIC'20 LOG ($RM = s, a, x$).

RF	$\mathcal{P}_{k=4}$	$\mathcal{P}_{\in R,k=4}$	$\mathcal{P}_{\notin R,k=4}$	BPIC'20	$\mathcal{P}_{k=3}$	$\mathcal{P}_{k=4}$	$\mathcal{P}_{k=5}$
I =none	230.25	180.63	348.43	I =none	**3.944**	2.606	2.382
$I = \bar{t}_R(\bar{c}_R)$	232.74	174.85	370.62	$I = \bar{t}_R$	3.973	**2.505**	**2.076**
$I = \overline{p}_4(\bar{c}_R)$	183.46	121.95	329.96	$I = \overline{p}_2$	3.976	2.587	2.322
$I = \overline{p}_{10}(\bar{c}_R)$	179.80	117.76	327.56	$I = \overline{p}_4$	3.980	2.529	2.174
$I = \overline{p}_{20}(\bar{c}_R)$	**177.52**	113.90	329.05	$I = \overline{p}_6$	3.978	2.537	2.173
$I = t_R(c_R)$	95.03	48.13	206.75	$I = t_R$	3.979	2.221	1.519

neither model CM_S nor CM_R, since *Request Payment* is exclusively followed by batching at *Payment Handled*. We then applied our implementation to create different inter-case features I for these segments and built different distance-to-batch models TM using I and trained corresponding RTP models RM. An attempt to compare our implementation to [6] failed because both logs did not contain start event recordings necessary for this method, resulting in a significant amount of empty inter-case features.

C. Results

Table II and Figs. 8, 9 and 10 summarize the results of different inter-case features I against the RF log and BPIC'20 log for the relevant prefix-lengths.

(1) As shown in Table II, we were able to improve prediction accuracy for both event logs and relevant k. For the RF log, we achieved a maximum error reduction of 23% for all cases and of 37% for batched cases using $I = \overline{p}_{20}(\bar{c}_R)$. For the BPIC'20 log, we achieved an error reduction of 13% using $I = \bar{t}_R$ for prefixes of $k = 5$. In Table II (left, RF log) we observe that our method mostly results in a reduced error for batched cases and results in a marginal error reduction of 6% for non-batched cases. For the distance-to-batch prediction by TM we find a MARE of 0.43 for the batched cases of the RF log and a MARE of 0.69 for the entire BPIC'20 log. The RTP models RM then show a MARE of 0.20 (batched RF) and of 0.73 (BPIC'20). Thus, the error in TM to predict the batching moment clearly has an impact on the accuracy of RM. When comparing the impact of different I on performance across logs (Table II left and right), we observe that the time distance $\bar{t}_R$ yields better performance in the case of frequent batching, i.e. short intervals (BPIC'20), while the partition distance $\overline{p}_n$ yields better performance in the case of large batch-intervals (RF), suggesting that a distance in time units might be more error prone when used for large batch-intervals. Finally, in Table II (right, BPIC'20 log) we observe larger error reductions for larger k. This can most likely be explained by the fact that the percentage of batched cases grows with $k = 3, 4, 5$ being 0%, 57% and 78%, respectively.

(2) Qualitatively reviewing predicted $\overline{y}$ over actual y remaining times for batched cases in RF, we see in Fig. 8 for $I = \varnothing$, 71% of cases are over-predicted and 29% under-predicted, resulting in two clusters on either side of the identity line. These clusters move closer to the identity line for $I = \overline{p}_{20}(\bar{c}_R)$ (Fig. 8 middle). However, several batched cases are still heavily under-predicted, which is most likely due to a wrong classification by model CM_S. For the ceiling $I = t_R(c_R)$ of actual time-to-batch (Fig. 8 right), the predictions align with the actual values. The same comparison for non-batched cases shows that both prior and our RTP models heavily over-predict non-batched cases.

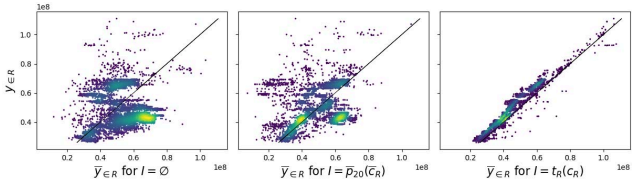


Fig. 8. Density scatter plot of $y_{\in R}$ and $\overline{y}_{\in R}$ using $I = \varnothing$, $I = \overline{p}_{20}(\bar{c}_R)$ and $I = t_R(c_R)$, and $RM = s, l, r$ for the RF log with $k = 4$.

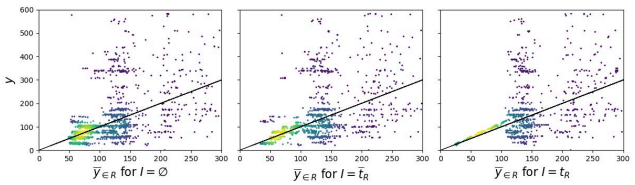


Fig. 9. Density scatter plot of y and $\overline{y}$ using $I = \varnothing$, $I = \bar{t}_R$ and $I = t_R$, and $RM = s, a, x$ for the BPIC'20 log with $k = 4$.

Qualitatively reviewing predicted $\overline{y}$ over actual y remaining times for all cases in BPIC'20, we see in Fig. 9 for $I = \varnothing$, 46% of cases are over-predicted and 54% under-predicted, resulting in three clusters, of which two overlapping the identity line and one above it. For $I = \bar{t}_R$, the predictions from the left-most cluster are better aligned with the actual values (Fig. 9 middle) and even more so for $I = t_R$ (Fig. 9 right). The other clusters appear almost identical across the three I and are most likely the 43% of cases that are only batched later in the process and do not have an inter-case feature yet.

Applying our own techniques of Sect. III, the OPS in Fig. 10 shows that the predictions of $I = \overline{p}_{20}(\bar{c}_R)$ form a clearer batching pattern than the predictions of $I = \varnothing$ (Fig. 2). Comparing the predicted final timestamps (Fig. 10 bottom) shows that (1) most under-predicted cases for the first batch moved closer to BM_1, (2) the mix of over- and under-predicted cases between BM_1 and BM_2 are separated and moved closer to their corresponding BM_i and (3) the set of over-predicted cases after BM_2 is slightly thinned out.

(3) Table III shows the results of the best I per log against $I = \varnothing$ across different RTP training methods (B, E, A). We were able to improve prediction results for the RF log across all methods with a maximum error reduction of 23% for method (s, l, r) and were able to obtain a minimum MAE of 163 days for method (c, l, x) vs 186 (baseline). For the BPIC'20 log, we were able to improve accuracy for 13 and 15 out of 18 methods for $k = 4$ and $k = 5$, respectively, with a maximum error reduction of 13% for method (s, a, x) and $k = 5$ and were able to obtain a minimum error of 1.95 days for method (c, l, r) and $k = 5$.

TABLE III
MAE OF BEST PERFORMING INTER-CASE FEATURES I ACROSS DIFFERENT PREDICTION METHODS (B, E, A).

Method w/ config. (B,E,A)		c,a,x	c,i,x	c,l,x	p,a,x	p,i,x	p,l,x	s,a,x	s,i,x	s,l,x	c,a,r	c,i,r	c,l,r	p,a,r	p,i,r	p,l,r	s,a,r	s,i,r	s,l,r
RF	$I=\varnothing$	196.50	202.64	191.77	202.92	197.90	200.27	200.26	199.40	186.36	200.73	202.15	226.38	201.89	204.98	225.47	201.87	206.04	*230.25*
$k=4$	$I=\overline{p}_{10}(\overline{c}_R)$	**170.94**	**172.79**	**162.83**	**174.78**	**170.82**	**170.66**	**195.84**	**198.71**	**172.65**	**172.56**	**169.58**	**178.49**	**177.00**	**171.76**	**178.30**	**168.54**	**181.12**	***177.52***
BPIC'20	$I=\varnothing$	2.608	2.545	2.490	2.662	**2.525**	**2.480**	*2.606*	2.485	2.523	2.762	2.678	2.544	2.789	**2.687**	**2.668**	2.792	**2.700**	2.841
$k=4$	$I=\overline{t}_R$	**2.573**	**2.533**	**2.462**	**2.587**	2.529	2.490	***2.505***	**2.436**	**2.511**	**2.739**	**2.599**	**2.536**	**2.780**	2.688	2.689	**2.777**	2.703	**2.839**
BPIC'20	$I=\varnothing$	2.325	2.121	2.057	2.438	**2.231**	**2.096**	*2.382*	2.157	2.002	2.471	2.213	1.997	2.365	**2.081**	2.158	2.448	2.189	2.173
$k=5$	$I=\overline{t}_R$	**2.084**	**2.113**	**2.031**	**2.293**	2.467	2.188	***2.076***	**2.111**	**1.989**	**2.198**	**2.089**	**1.952**	**2.191**	2.114	**2.043**	**2.219**	**2.115**	**2.160**

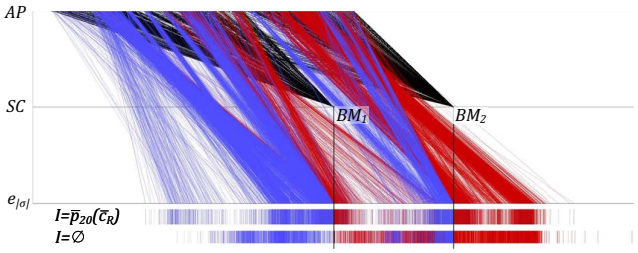


Fig. 10. OPS of RF log for $S=(AP,SC)$ and $k=4$, using $RM=s,l,r$ and $I=\overline{p}_{20}(\overline{c}_R)$, compared to $I=\varnothing$ (Fig. 2).

VI. CONCLUSION

We introduced visual analytics techniques based on the performance spectrum [19] to qualitatively analyze the nature of remaining time prediction errors. Our evaluation showed that engineering inter-case features that give a case context for dynamics in other cases, e.g, batching, allows to significantly reduce the prediction error by up to 37%. This cycle of qualitative evaluation and feature engineering makes a first step to a *qualitative feedback loop* in effectively improving remaining time prediction models.

Qualitatively evaluating our own experimental results shows that batching behavior is both quantitatively and qualitatively picked up with our features. Though, a large amount of batched cases remain significantly under- and over-predicted, i.e., batches are not accurately captured, non-batched cases are heavily over-predicted, and we only improve performance for cases that have batching in the near future. The performance improvement relies on the accuracy of the inter-case feature and to what extent it is clearly present in the data set, i.e., the nature of the process. Next steps are to improve the incorporation of batching in remaining time prediction, but also explore other process mechanisms and -entities that cause breaches in ideal process behavior. A next improvement cycle should for instance focus on inter-case dynamics in non-batched cases, e.g., due to FIFO behavior or overtaking.

REFERENCES

[1] M. Dumas, M. La Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management*. Springer, 2013.

[2] I. Verenich, M. Dumas, M. La Rosa, F. M. Maggi, and I. Teinemaa, "Survey and Cross-Benchmark Comparison of Remaining Time Prediction Methods in Business Process Monitoring," *ACM Trans. Intell. Syst. Technol.*, vol. 10, July 2019.

[3] V. Denisov, D. Fahland, and W. M. P. van der Aalst, "Unbiased, Fine-Grained Description of Processes Performance from Event Data," in *BPM 2018*, vol. 11080 of *LNCS*, pp. 139–157, Springer, 2018.

[4] E. L. Klijn and D. Fahland, "Performance Mining for Batch Processing Using the Performance Spectrum," *LNBIP*, vol. 362, pp. 172–185, 2019.

[5] A. Senderovich, C. Di Francescomarino, C. Ghidini, K. Jorbina, and F. Maggi, "Intra and Inter-case Features in Predictive Process Monitoring: A Tale of Two Dimensions," in *BPM*, pp. 306–323, Springer, 2017.

[6] A. Senderovich, C. Di Francescomarino, and F. M. Maggi, "From Knowledge-Driven to Data-Driven Inter-Case Feature Encoding in Predictive Process Monitoring," *Information Systems*, vol. 84, pp. 255–264, 2019.

[7] A. Senderovich, J. Beck, A. Gal, and M. Weidlich, "Congestion Graphs for Automated Time Predictions," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 4854–4861, 07 2019.

[8] 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.

[9] A. Rogge-Solti and M. Weske, "Prediction of Business Process Durations Using Non-Markovian Stochastic Petri Nets," *Information Systems*, vol. 54, pp. 1–14, 2015.

[10] A. Senderovich, M. Weidlich, A. Gal, and A. Mandelbaum, "Queue Mining for Delay Prediction in Multi-Class Service Processes," *Information Systems*, vol. 53, pp. 278–295, 2015.

[11] 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.

[12] N. Tax, I. Verenich, M. La Rosa, and M. Dumas, "Predictive Business Process Monitoring with LSTM Neural Networks," in *CAiSE*, pp. 477–492, Springer, 2017.

[13] A. Pika, W. M. P. van der Aalst, C. J. Fidge, A. H. M. ter Hofstede, and M. T. Wynn, "Predicting Deadline Transgressions Using Event Logs," in *Business Process Management Workshops*, pp. 211–216, Springer, 2013.

[14] I. Verenich, H. Nguyen, M. La Rosa, and M. Dumas, "White-Box Prediction of Process Performance Indicators via Flow Analysis," in *ICSSP*, pp. 85–94, Association for Computing Machinery, 2017.

[15] W. M. P. van der Aalst, *Process Mining: Data Science in Action*. Springer, 2 ed., 2016.

[16] D. Sarkar, R. Bali, and T. Sharma, *Practical Machine Learning with Python*. Apress, 1st ed., 2017.

[17] R. Conforti, M. de Leoni, M. La Rosa, W. M. 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.

[18] E. L. Klijn, "Explainable Remaining Time Prediction for Business Processes," Master's thesis, Eindhoven University of Technology, 2020.

[19] V. Denisov, E. Belkina, D. Fahland, and W. M. P. van der Aalst, "The Performance Spectrum Miner: Visual Analytics for Fine-Grained Performance Analysis of Processes," in *BPM 2018 Demos*, vol. 2196 of *CEUR Workshop Proceedings*, pp. 96–100, CEUR-WS.org, 2018.

[20] P. J. Brockwell and R. A. Davis, *Introduction to Time Series and Forecasting*. Springer, 3rd ed., 2016.

[21] M. De Leoni and F. Mannhardt, "Road Traffic Fine Management Process." Dataset, 2015. https://doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5.

[22] B. F. van Dongen, "BPI Challenge 2020: Domestic Declarations." Dataset, 2020. https://doi.org/10.4121/uuid:52fb97d4-4588-43c9-9d04-3604d4613b51.

[23] I. Teinemaa, M. Dumas, M. La Rosa, and F. M. Maggi, "Outcome-Oriented Predictive Process Monitoring: Review and Benchmark," *ACM TKDD*, vol. 13, no. 2, pp. 1–57, 2019.

Time-aware Concept Drift Detection Using the Earth Mover's Distance

Tobias Brockhoff[0000-0002-6593-9444], Merih Seran Uysal[0000-0003-1115-6601],
Wil M.P. van der Aalst[0000-0002-0955-6940]
Process and Data Science Group (PADS)
Computer Sience 9, RWTH Aachen University, Aachen, Germany
{brockhoff, uysal, wvdaalst}@pads.rwth-aachen.de

***Abstract*—Modern business processes are embedded in a complex environment and, thus, subjected to continuous changes. While current approaches focus on the control flow only, additional perspectives, such as time, are neglected. In this paper, we investigate a more general concept drift detection framework that is based on the Earth Mover's Distance. Our approach is flexible in terms of incorporating additional perspectives thanks to the capability of defining custom feature representations, as well as expressive feature similarity measures. We demonstrate the former by incorporating the time perspective using both a time-binning-based trace descriptor and a suitable similarity measure that considers time and control flow. We evaluate the resulting sliding window detector on different types of control-flow and time drifts, and holistic drifts involving multiple perspectives.**

***Index Terms*—Process Mining, Concept Drift Detection, Earth Mover's Distance**

I. Introduction

Due to changing conditions in business environments, business processes are continuously evolving and are, thus, seldomly in a steady state. Therefore, flexibility, supported by efficient and holistic change detection and management approaches, is an important competitive advantage and can even be necessary to persist in a competitive environment.

A major challenge for detecting concept drift is its multifactorial causes and effects. For example, new regulations change the control flow, the workload is adapted to changing market demand/price, the organization of the individual work changes over time, or new employees temporarily reduce the performance of specific parts of the process. Hence, concept drift detection methods that allow for different perspectives onto the process are needed.

To this end, this paper proposes a drift detection method based on the Earth Mover's Distance (EMD) [1] which allows us to incorporate different perspectives. Introduced in the computer vision domain as a similarity measure matching human perception well, EMD has recently gained interest in the process mining community as a method for conformance checking on stochastic languages [2]. Given two distributions and a measure of pairwise feature similarity, EMD describes the minimum effort that is required to transform one distribution into the other. Its intuitive interpretation and flexibility in terms of specifying a feature distribution and similarity measure makes EMD a general comparison method that has been applied in many different domains.

In contrast to most of the existing work which solely focuses on control-flow level drifts by means of hypothesis tests on control flow feature distributions, we propose to exploit the flexibility of EMD to address other manifestations of drift. For instance, from the resource perspective, the frequency that a certain resource executes a sequence of activities might change, while these activities are still handled by other resources. Moreover, the control-flow level drift detection does not consider *time* which is a major driver for key performance indicators. For example, an increasing case arrival rate will ultimately impact the sojourn times if the available resources become overloaded. Furthermore, the service times of certain activities, as a proxy for case complexity, can determine how the case will be further handled and, therefore, impact the general control flow. These use cases require methods that extend the general control-flow perspective and also consider time, resources or combinations of the former perspectives yielding a broader or even holistic view on potential changes.

Using EMD we demonstrate how a *time* perspective can be incorporated into concept drift detection, e.g., activity service and sojourn times. To achieve this, we use both a time-aware trace descriptor and a suitable similarity measure.

Our contributions are as follows: First, we introduce a sliding window-based concept drift detection approach that uses stochastic languages to represent the windows. Then, we use EMD to compare two windows with each other and detect a (possible) concept drift between them. Moreover, we demonstrate how the basic detection approach can be modified in order to incorporate time features.

The remainder of the paper is organized as follows: We review the related work in Section II. Then, Section IV-A introduces EMD for stochastic languages and Section IV-B demonstrates how time can be incorporated into the detection approach. After presenting the sliding windows method in Section IV-C, we evaluate our approach in Section V. Finally, we conclude our paper and give an outlook on future work in Section VI.

We thank the Alexander von Humboldt (AvH) Stiftung for supporting our research. Funded by the Deutsche Forschungsgemeinschaft(DFG, German Research Foundation) under Germany's Excellence Strategy — EXC-2023 Internet of Production 390621612.

978-1-7281-9832-3/20/$31.00 ©2020 IEEE
DOI 10.1109/ICPM49681.2020.00016

II. Related Work

Originally, the topic of *concept drift* originated from the fields of data mining and machine learning where it describes a change of a target concept [3], [4] dependent on a hidden context. As summarized in [5], there are three major approaches for detecting concept drift. Approaches based on *sequential analysis* incrementally compute and maintain a test statistic on distribution changes of the input data, e.g., whether the probability ratio of certain subsequences changes [6]. Methods based on *statistical process control* maintain a model of the process and detect change points by monitoring prediction quality metrics [7], [8]. Finally, *window-based drift detectors* compare the empirical data distributions between two data windows at different periods in time. In this context, differences between the distributions can be assessed directly by statistical tests [9], assuming equal distributions as the null hypothesis, or by significant changes of information theoretic measures, e.g., Kullback-Leibler divergence [10].

In process mining, *window-based concept drift detectors* are the most prominent type of detector. The first approach specifically tailored to business processes was proposed in [11]. The authors apply two fixed-size sliding windows representing traces by feature vectors proposing two global and two local features, respectively. Differences between the feature vector distributions are then assessed by a hypothesis test. Finally, sliding the windows over the log yields a p-value plot that can be used for visual concept drift detection. A more detailed analysis of this approach can be found in [12]. Moreover, using improved windowing techniques, this work was automatized and extend in [13] to deal with gradual drift and multi-order dynamics, i.e., multiple changes of different granularities. A similar window-based approach representing traces by means of a trace abstraction in terms of so-called *runs* was proposed in [14]. Based on the classical α-relations [15], distributions over trace abstractions are computed and compared using a hypothesis test. The idea of using α-relations for drift detection was also extended to drift detection and characterization in event streams [16], [17]. The authors in [18] propose to compute pair-wise intra-trace activity distances over a reference window where distance interval changes indicate change points. A window-based approach that chooses a different window representation based on convex polyhedra over the Parikh vector [19] can be found in [20]. Finally, window comparison using the canberra distance on the dependency relations of the heuristic miner was proposed in [21].

In contrast to the window-based approaches, a *sequential analysis* based approach, that maintains statistics on whether directly-follows and eventually-follows relations always, never, or sometimes hold within sequential sections of the process, was introduced in [22]. Nevertheless, all these approaches only consider the control flow level.

Clustering is another approach for detecting changes. The authors in [23] propose to enhance the trace representation by a time dimension by adding the start timestamp of the trace. Although this explicitly incorporates time into the representation, time information of individual activities is not considered. A more general framework for trace clustering based change point detection is introduced in [24]. Traces within a chosen window are clustered using the Markov Cluster algorithm and the trace of the difference matrix between two cluster matrices is proposed as a window similarity measure. The downside of this approach is that control-flow features and time information are not considered by the authors. Moreover, cluster differences were only evaluated visually.

III. Prelimiaries

Before proposing our novel concept drift detection method, this section introduces the basic concepts needed in Section IV.

a) Event Log: Let Σ_A denote a finite alphabet of activities and let Σ_V be the set of (countable infinite) additional activity descriptor values that define a view on the process, e.g., $\Sigma_V = \mathbb{N}$ for temporal information based on binning (cf. Section IV-B). Each event can then be described by an activity and its additional descriptor value yielding an alphabet $\Sigma = \Sigma_A \times \Sigma_V$. Given a descriptor $e = (a, v) \in \Sigma$, $e_a = a$, $e_v = v$ denote the corresponding activity and view value, respectively. Subsequently, the set Σ^* of all finite words over Σ describes the possible *trace variants* of the process. For a trace $\sigma = (e_1, e_2, \ldots, e_n) \in \Sigma^n$, we denote the i-th element by $\sigma_i = e_i$ and the subsequence $(e_i, \ldots e_j)$ by $\sigma[i, j], i \leq j$. Using the potentially extended alphabet, the input of our method is an event log that is defined as follows:

Definition 1 (Event Log)**.** *Let Σ denote a (countable infinite) alphabet. An* event log *is a finite multiset of traces $E \colon \Sigma^* \to \mathbb{N}$.*

For example, $E = [\langle (a,1),(b,4)\rangle^7, \langle (a,4),(b,2)\rangle^3, \langle (b,2),(a,4)\rangle^1]$ is an event log containing 10 traces comprising an additional integer descriptor.

b) Stochastic Language: Stochastic languages can be used to represent collections of traces. In addition to the *trace variants* that are present, a *stochastic language* also captures the likelihood of a variant. Formally, it is defined as a function that assigns a probability to each trace.

Definition 2. *Given an alphabet Σ, $f \colon \Sigma^* \to [0, 1]$ is a stochastic language iff $\Sigma_{t \in \Sigma^*} f(t) = 1$.*

Each multiset of traces, e.g., an event log, can be transformed into a *stochastic language* by normalization. For instance, the preceding example event log can be normalized to $[\langle (a,1),(b,4)\rangle^{\frac{6}{10}}, \langle (a,4),(b,2)\rangle^{\frac{3}{10}}, \langle (b,2),(a,4)\rangle^{\frac{1}{10}}]$. The support of a *stochastic language* f will be denoted by $S_f = \{t \in \Sigma^* | f(t) > 0\}$. In the following, we will consider *finite stochastic languages*, i.e., *stochastic languages* with finite support.

IV. Methods

In this section, we propose an approach for time-aware concept drift detection. To this end, we first introduce the Earth

Mover's Distance (EMD) as a comparison method for probability distributions. Regarding the importance of incoporating additional perspectives for holistic change point detection, using the proposed method, we demonstrate how the detector can naturally be extended to consider time. Finally, we briefly describe the sliding window approach that was used to make the method applicable for concept drift detection.

A. Concept Drift Detection based on EMD

The Earth Mover's Distance (EMD) is a distance-based similarity measure for the comparison of probability distributions. Intuitively, it considers one distribution as piles of earth and the other as a set of holes. Given a ground distance function that describes the effort of moving earth from a certain pile to a specific hole, EMD is the minimum-cost flow required to move earth from hills to holes.

a) Trace Distance: Before formally defining EMD as a distance measure for stochastic languages, we introduce the notion of a trace distance [2]. A trace distance $\delta\colon \Sigma^* \times \Sigma^* \to [0,1]$ describes the dissimilarity of any two given traces. It is required to be symmetric and to satisfy the identity of indiscernibles porperty [2]. Besides, it is desirable that δ additionally satisfies the triangle inequality, i.e., $\forall t, t', t'' \in \Sigma^*\colon \delta(t,t') + \delta(t',t'') \leq \delta(t,t'')$, so that δ, and therefore also EMD, become a metric. Nevertheless, as detailed in Section IV-B, some trace distance candidates might only empirically satisfy the triangle inequality for most inputs.

b) EMD for Stochastic Languages: Given a trace distance function, EMD as a measure for comparing stochastic languages has been proposed in [2]. In general, EMD as a similarity measure between probability distributions unites two desirable properties. On the one hand, it is frequency-aware in the sense that it considers the magnitude of discovered differences. On the other hand, the notion of difference is determined by the trace distance, and, thus, different distance function can express different perceptions of similarity. In the context of processes, we use this property to open different perspectives on the process. Formally, in the context of finite stochastic languages f, g, EMD between f and g can be defined by the following linear program:

Definition 3 (Earth Mover's Distance)**.** *Let f, g be finite stochastic languages over the alphabet Σ and let δ be a trace distance function. The Earth Mover's Distance between f and g is defined by:*

$$EMD(f,g) = \textit{minimize} \quad \sum_{t \in S_f} \sum_{u \in S_g} x_{tu} \cdot \delta(t,u)$$

$$\textit{s.t.} \qquad \forall t \in S_f : \sum_{u \in S_g} x_{tu} = f(t) \quad \textit{(Source)}$$

$$\forall u \in S_g : \sum_{t \in S_f} x_{tu} = g(u) \quad \textit{(Target)}$$

$$\forall t \in S_f \, \forall u \in S_g : \quad x_{tu} \geq 0 \quad \textit{(Non-negativity)}.$$

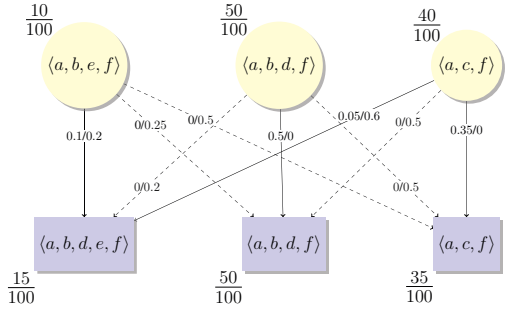


Fig. 1: Optimal cost flow for $EMD(f_1, f_2)$ for the stochastic languages f_1, f_2. Edge inscriptions $x^*_{tu}/\delta(t,u)$ show the optimal flow x^*_{tu} and the trace distance value $\delta(t,u)$ for all variants $t \in S_{f_1}, u \in S_{f_2}$. Zero flow edges are depicted by dashed lines. Even though we have $S_{f_1} \neq S_{f_2}$, $EMD(f_1, f_2) = 0.05$ shows that the difference concerns infrequent behavior.

While the last constraints ensure non-negative flows, the source constraints assert that the entire probability mass for each trace of f is moved to traces in g. Likewise, the target constraints limit the flows to g according to its probability mass distribution. Furthermore, the objective function ensures a minimal effort flow. Note that, as we consider finite stochastic languages, the number of variables and constraints is finite, and, thus, the linear program is well-defined. Moreover, the linear program is feasible due to the equality of the total weight of f and g as stochastic languages are inherently normalized. Besides, normalization and a metric as trace distance would make EMD on stochastic languages a metric too [1].

To detect control flow changes between two stochastic languages, we can for example use the post-normalized Levenshtein distance, where in a second step the computed distance value is normalized by the maximum length of the two considered sequences, as trace distance function. For instance, consider the stochastic languages $f_1 = [\langle a,b,d,f\rangle \frac{50}{100}, \langle a,c,f\rangle \frac{40}{100}, \langle a,b,e,f\rangle \frac{10}{100}]$, $f_2 = [\langle a,b,d,f\rangle \frac{50}{100}, \langle a,c,f\rangle \frac{35}{100}, \langle a,b,d,e,f\rangle \frac{15}{100}]$ and $f_3 = [\langle a,b,d,f\rangle \frac{20}{100}, \langle a,c,f\rangle \frac{70}{100}, \langle a,b,e,f\rangle \frac{10}{100}]$ where f_1 represents the basic process behavior. While in f_2 an infrequent new variant where both d and e can occur is introduced replacing another infrequent variant, in f_3 the most frequent variant changes from $\langle a,b,d,f\rangle$ to $\langle a,c,f\rangle$. Using the (post)normalized Levenshtein distance, we obtain $EMD(f_1, f_2) = 0.05$ and $EMD(f_1, f_3) = 0.15$ which well reflects the major behavior change in f_3. The example solution to the linear program for $EMD(f_1, f_2)$ is depicted in Figure 1. It shows that, on the one hand, the costs are caused by transferring a probability mass of 0.1 between the two infrequent variants $\langle a,b,e,f\rangle$ and $\langle a,b,d,e,f\rangle$, which have a post-normalized trace distance of 0.2. On the other hand, a slightly reduced likelihood of $\langle a,c,f\rangle$ in f_2 causes additional costs of $0.05 \cdot 0.6 = 0.03$. Using EMD for stochastic language comparison enables us to focus on trace properties by means of the trace distance function. Moreover, it also allows us to consider the frequency of observed patterns.

B. Time-Aware Concept Drift Detection

In this section, we propose to extend concept drift detection by adding a time perspective, i.e., sojourn times or activity service times. To this end, we exploit the flexibility of EMD and introduce a time-aware ground distance. We first bin the observed times and then apply a weighted Levenshtein distance variant in order to obtain a control flow and time-aware comparison method.

a) Binning: We apply time binning in order to abstract similar time related behavior and to make the weighted Levenshtein distance applicable. To this end, depending on the chosen time perspective, first, we collect the observed activity service or sojourn times. As time features might differ significantly between different activities, we locally evaluate the following two clustering approaches, i.e., we compute the statistics per activity while taking the complete log into account. Following the Pareto Principle that 80% of the interesting behavior is caused by 20% of the cases, we propose to determine the bins by a percentile-based clustering based on the times observed for each individual activity. The bin edges are thereby chosen according to a given list of percentiles. Regarding multimodal time distributions we propose to use the k-means++ [25] as a data-driven clustering approach. In the 1-dimensional time setting its objective is theoretically well-founded being equivalent to minimizing the within-class variance [26]. A major advantage is that this approach can deal with different cluster sizes among different activities. By applying these binning techniques, we discretize time-related information into k bins and obtain traces over an alphabet $\Sigma_T = \Sigma_A \times \{0, \dots, k-1\}$ with the bin index as an additional descriptor.

b) Time-Aware Trace Distance: Given two traces $\sigma, \sigma' \in \Sigma_T^*$ over the binned alphabet with k bins, we propose to use a variant of the weighted Levenshtein distance δ_l [27] which splits the total operation costs into control-flow and time costs, respectively. To this end, on the control-flow level, we define binary costs $c_r^f \colon \Sigma_A \times \Sigma_A \to \mathbb{R}^{\geq 0}$ for renaming and unary costs $c_{\text{id}}^f \colon \Sigma_A \to \mathbb{R}^{\geq 0}$ for insertion and deletion. Likewise, on the time level, we use binary costs $c_{\text{mr}}^t \colon \{0, \dots, k-1\} \times \{0, \dots, k-1\} \to \mathbb{R}^{\geq 0}$, for matching and renaming activities considering different time bins and unary costs $c_{\text{id}}^t \colon \{0, \dots, k-1\} \to \mathbb{R}^{\geq 0}$ for insertion and deletion. The resulting trace distance can then be defined by the following recursive relation:

$$\delta_l(\sigma[1,i],\sigma'[1,j])= \min \begin{cases} \delta_l(\sigma[1,i-1],\sigma'[1,j-1]) + c_{\text{mr}}^t((\sigma_i)_v,(\sigma'_j)_v) & \text{if}(\sigma_i)_a=(\sigma'_j)_a \\ \delta_l(\sigma[1,i-1],\sigma'[1,j-1]) + & \\ \quad c_r^f((\sigma_i)_a,(\sigma'_j)_a) + c_{\text{mr}}^t((\sigma_i)_v,(\sigma'_j)_v) & \text{(Rename)} \\ \delta_l(\sigma[1,i],\sigma'[1,j-1]) + c_{\text{id}}^f((\sigma'_j)_a) + c_{\text{id}}^t((\sigma'_j)_v) & \text{(Insert)} \\ \delta_l(\sigma[1,i-1],\sigma'[1,j]) + c_{\text{id}}^f((\sigma_i)_a) + c_{\text{id}}^t((\sigma_i)_v) & \text{(Delete)}\,. \end{cases} \tag{1}$$

Regarding a concrete instantiation of the cost functions, we need to ensure that it respects the inter-dependencies between the edit operations to preserve the intuitive interpretability of the distance value which is stated in the following axiom.

Axiom 1 (Levenshtein-based Trace Distance Interpretability)**.** *Levenshtein-based trace distances that consider time information should fulfill the following properties:*

1. *Renaming is preferable over deletion and insertion*
2. *Matching events under a large time difference is less expensive than*
   (a) *deletion followed by insertion*
   (b) *renaming events with different activity labels but the same time information.*

Moreover, since we will use δ_l in a sliding-window framework, we want δ_l to be symmetric so that traces from the left and right window are treated equally. For example, consider three subsequent windows each containing just a single trace variant, namely σ^1, σ^2 and σ^3, such that the first and third window are equal, i.e., $\sigma^1 = \sigma^3$. Intuitively, differences and, hence, potential drift points between the first and the second, or between the second and the third window should be scored equally, i.e., $\delta_l(\sigma, \sigma') = \delta_l(\sigma', \sigma'')$. Therefore, we use the same function for the insertion and deletion cost to make δ_l symmetric.

Considering the interpretation axioms, let $a, a' \in \Sigma_A, a \neq a'$, $t, t' \in \{0, \dots, k-1\}$. We use fixed control-flow penalties for renaming, insertion, and deletion, i.e., $c_r^f(a, a') = f_r = 1 = f_{\text{id}} = c_{\text{id}}^f(a)$, costs linear in the bin distance for matching and renaming w.r.t. time, i.e., $c_m^t(t, t') = c_r^t(t, t') = |t - t'|$, and cost linear in the bin index for insertion and deletion, i.e., $c_{\text{id}}^t(t) = t$. Even though time bins of disjoint activities are not necessarily comparable w.r.t. absolute time, this choice naturally enforces Axiom 1.2(b). For example, consider the costs and traces $f_r = 2, f_{\text{id}} = 2$, $\sigma = \langle (a,1),(b,4) \rangle$, $\sigma' = \langle (a,4),(b,2) \rangle$ and $\sigma'' = \langle (b,2),(a,4) \rangle$. If renaming can only be applied to different activity labels, is not per se expensive and does not respect time differences, we obtain $3 + 2 = \delta_l(\sigma, \sigma') \geq \delta_l(\sigma, \sigma'') = 2 + 2$ although renaming is required twice in the latter case. This contradicts the intuitive perception of similarity. Likewise, for high renaming costs, that are independent of the time difference, renaming operations would dominate EMD, therefore, rendering it less sensitive to time behavior changes. In addition to a general operation penalty, we also consider the time bin for insertion and deletion operations enforcing Axiom 1.2(a).

In order to account for different trace lengths, we post-normalize the distance by the maximum trace lengths, i.e., $\bar{\delta}_l(\sigma, \sigma') = \frac{\delta_l(\sigma,\sigma')}{\max(|\sigma|,|\sigma'|)}, \sigma, \sigma' \in \Sigma^*$. Even though δ_l is a metric, normalization might violate the triangle inequality for Levenshtein based edit distances [28]. Accordingly, $\bar{\delta}_l$ is not necessarily a metric. Nevertheless, the triangle inequality and thus metric properties emprically hold for most of the descriptors considered in our evaluation.

C. Sliding Window

In order to detect concept drift in an event log, we embed our approach into a sliding window framework. Let $(t_1, t_2, \dots, t_n) \in (\Sigma^*)^n$ denote an ordering of the traces

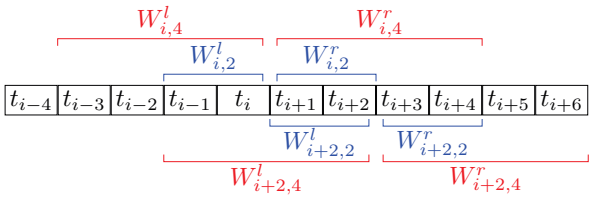


Fig. 2: Multi-scale sliding window. At each scale, viz. window size 2 and 4, we maintain a left and right window. After comparison, both windows are shifted by a stride size of 2.

by the timestamp of the first event. As depicted in Figure 2, given an index i, $W^l_{i,w} = (t_{i-w+1}, t_{i-w}, \ldots, t_i)$ and $W^r_{i,w} = (t_{i+1}, t_{i+2}, \ldots, t_w)$ denote the left and right window, respectively. We compute the corresponding trace descriptors for each window pair $W^l_{i,w}$ and $W^r_{i,w}$ and transform the resulting representations into stochastic languages f^l_i and f^l_r by normalization. Afterwards, we compare f^l_i and f^l_r using EMD yielding the dissimilarity score $EMD(f^l_i, f^r_i)$. Finally, the windows are shifted by a user-defined stride size. Increasing the stride size, thereby trades localization accuracy for efficiency. We plot the final EMD values to detect process changes. Similar to the work in [29], we use different-sized windows in order to show and validate drift on multiple scales.

V. Evaluation

In this section, we demonstrate the capability of the proposed method to detect control-flow drifts, time drifts, and control-flow drifts which can only be detected by a holistic approach. To this end, we created multiple artificial event logs from variants of the model used in [11]. Figure 3 depicts the adapted model, which models the process of handling health insurance claims, in BPMN notation. Arriving claims are registered and classified into low and high claims. Besides, a questionnaire is sent to the claimant which might be answered. Each claim undergoes several checks which either lead to rejection or eventually to acceptance. In contrast to the original model, our model adapts the efficient *reject-if-check-is-unsuccessful* pattern for low and high claims. Furthermore, our model introduces refunding activities before the claimant is notified. After checking, every claimant receives a documented notification and can be called additionally. Finally, the case is archived while some cases are also controlled in more detail. In order to assign the tasks to resources, we maintain a resource pool and randomly choose free resource therefrom. Following natural behavior, we sample exponentially distributed inter-case arrival times while the service times of human executed activities are normal distributed.

A. Implementation

We implemented the methods presented in Section IV as a plugin in ProM. The plugin allows to specify multiple sliding windows of different window and stride sizes. Moreover, it supports the Levenshtein distance and the weighted Levenshtein distance for time-binning based trace descriptors, that we proposed in Section IV-B, as trace distance functions. Regarding the time perspective, the user can choose between service and sojourn time-aware concept drift detection. In this case, the time bins are computed according to user-defined percentiles for the percentile-based bin edge computation or a given number of clusters for the data-driven approach using k-means. In order to efficiently compute EMD, we implemented an Exterior Point Simplex method described in [30], [31], which is tailored to the underlying optimal-cost-flow problem.

B. Control-Flow Drift

In order to demonstrate the ability of our approach to detect control-flow drifts, we apply it to the log used in [11]. The log contains different types of control-flow drift with a change every 1200 traces. Figure 4a shows the output of our plugin for different window sizes using EMD with Levenshtein distance for window comparison. The bottom panel shows a heatmap of EMD values w.r.t. the indices of the traces, ordered by the timestamp of the first event, and the window sizes. Given that different window sizes capture changes of different frequencies, this spectrum-inspired visualization can help to localize changes in the time and frequency domain. The upper panel shows a lineplot of the EMD values for an interactively selectable window size.

First, as depicted by the dotted lines, we note that all model changes can be detected, although the inherently high-dimensional trace descriptor requires large windows sizes to capture the process behavior. As can be seen for window size 200, the significantly higher variance indicates that the sample size is insufficient to fully represent the underlying distribution. In this case, larger windows are needed to verify potential drift points. Moreover, it can be seen that EMD naturally describes the extend to which the process is affected by the change. To this end, consider the highlighted EMD value difference between the second and third peak. While the pre-change model for the second peak contains an or-relation between 'By Phone', 'E-Mail', and 'Post', as well as the option to skip everything, the change removes the latter option. The low EMD values thereby indicate that relatively few cases are affected, i.e., the notification has been skipped for these cases. In contrast, the third peak results from introducing an exclusive choice between the former activities. Considerably larger EMD values show that many cases are affected, i.e., before the change claimants were often notified via multiple channels.

Although the currently implemented window descriptor based on stochastic languages, which consider entire traces, can be effectively high-dimensional, an advantage is that we can detect long-term dependencies. In order to show this, we introduce a controlling activity scrutinizing a limited subset of the recently archived cases. Initially, given two low and high claim cases, the probability that the high claim case is selected for additional controlling is 40%. After 2000 cases this probability changes to 20%, following the idea that high claim cases have been handled more carefully, and therefore less controlling is required. This introduces a long-term dependency between the initial check claim sequence

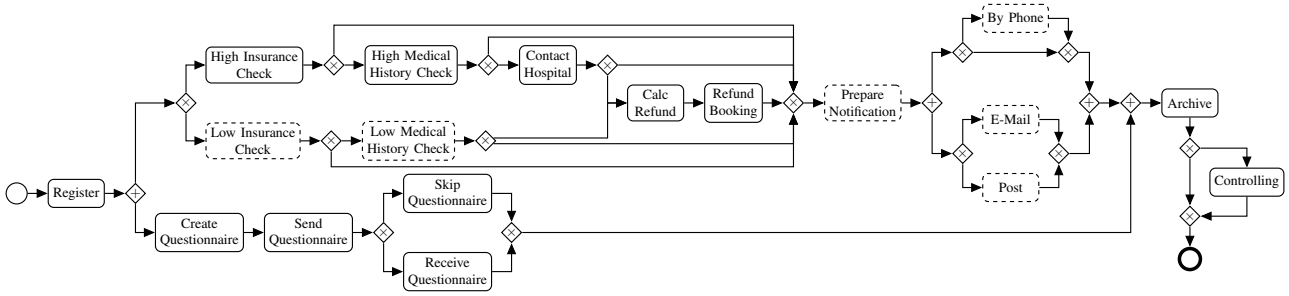
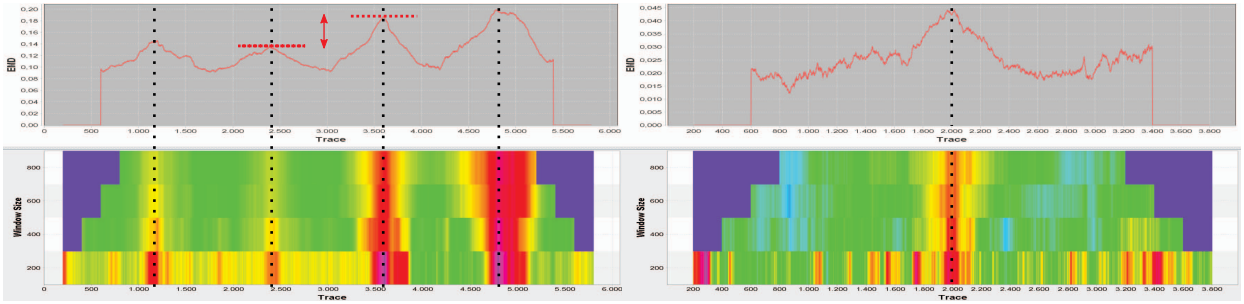


Fig. 3: The adapted insurance claim model from [11] showing a process for handling low and high insurance claims. Dashed lines show the activities that are affected by the filtering applied in the time-induced control flow drift scenario.



(a) Different control-flow drifts (stride size 2).

(b) Long-term dependency drift (stride size 2).

Fig. 4: Drift detection on the control-flow level using the Levenshtein distance. It shows the spectrum over window sizes 200, 400, 600, 800 and selectively EMD for window size 600.

and the controlling activity which is further extended by the introduction of the refunding activities (cf. Figure 3). As depicted in Figure 4b, the drift can be detected which would not be possible using local statistics, e.g., a window around each activity if the window size is too small [11]. Intuitively, such drift scenarios are relevant for activities whose number of occurrences is limited by the resource availability and, therefore, drifts do not necessarily affect the follows frequencies of directly preceding activities.

C. Time Drift

In order to assess the capability to detect time drift, we inject changes of the case arrival rate. Using an initial base inter-arrival time of 45 minutes, we created a log with sudden arrival rate drifts to 60, 45, 30, 45, 25, and 45 minutes every 2000 traces, respectively. In order to exclude the possibility of inter-arrival-time-induced control-flow drifts, we filter the *questionnaire-related activities*. Due to the increased sojourn times, more questionnaires are received back which causes a control-flow drift.

Regarding the parameter setting of f_{id} and f_r, preceding experiments showed that $f_{\text{id}} = f_r = 1$ works well. In general, we could see that the output of our method is not very sensitive to this setting for reasonably small values. Furthermore, we apply the data-driven bin computation using k-means, which generally performed well, with $k = 3$. This follows the intuition to have a bin for slow, moderate, and fast behavior.

As depicted by the dotted blue lines in Figure 5a, according to our method an increasing inter-arrival time does not significantly affect the sojourn times. During periods of baseline arrival rate, the resources are sufficient to handle each case immediately. Therefore, reducing the arrival rate does not affect the immediate reaction, and hence the sojourn times do not change. In contrast, decreasing the inter-arrival times increases the sojourn times and a drift is detected. The peaks corresponding to the onsets of arrival rate drifts are highlighted by dotted black lines. In addition to the ground truth arrival rate changes, we observe two additional peaks, depicted by the densely dotted lines, which might be attributed to the variance of the arrival rate and activity service times. Moreover, larger EMD values for shorter inter-arrival times show that EMD is a measure for the impact of the drift. Regarding our definition of the time-aware trace distance function δ_l, this experiment shows that incorporating the time bins into the costs for insertion and removal causes a shift of the baseline costs, as indicated by the dashed red line, so that the EMD values are generally larger during periods of faster case arrival. Given two traces for which the optimal distance according to δ_l requires deletion and insertion, a low arrival rate and, therefore, low sojourn times and bin indices yield a comparatively smaller distance value than a high arrival rate and consequently higher bin indices due to higher costs for insertion and deletion. Hence, the costs arising from imperfectly matching windows are generally higher. Since the baseline shifts accumulate,

we can, to some extend, also observe ongoing gradual drifts towards increasing sojourn times during the periods of arrival rate change.

D. Time-induced Control Flow Drift

Finally, we investigate a drift scenario that requires a holistic approach combining the time and control perspective. To this end, we distinguish between low claims with an easy medical history and those with a difficult history. In the model, this difference reflects in the service time of 'Low Medical History Check' with an increased service time for more difficult cases. The service times for each group are modeled by normal distributions where the probability of a low claim being difficult is 20%. The service times sampled from the resulting Gaussian mixture model are depicted in Figure 6. Assuming that the resources for calling claimants are limited and therefore only a fixed percentage of the claimants can be called, we introduce a drift by changing the individual probabilities for notification by phone for easy and difficult cases. While initially the probability is 50% for both degrees of difficulty, after the drift, that is introduced after 2500 cases, difficult cases are certainly called and easy cases only with a probability of approximately 40%. Note that in order to detect this drift, time has to be considered since the general control-flow on the activity level is not affected.

Detecting the aforementioned concept drift requires us to focus on the low claim check activities and the subsequent notification as there is only a small impact on the complete process. Therefore, we filter the generated log, containing 5000 traces, for low claim cases by only considering claims that experience the 'Low Insurance Check'. Note that cases that are rejected in this step are still considered, which slightly broadens the focus. Furthermore, we only focus on checking and notification related activities depicted by dotted lines in Figure 3. As before, we apply data-driven bin computation with $k = 3$. Figure 5b shows the output for the filtered log. The dotted line indicates a drift that occurs after half of the traces, which fits the time of the actual drift. The magnitude of EMD change also gives evidence that this drift only slightly affects the process, i.e., only a few claims are concerned. Moreover, the high variance and large EMD values that can be seen in the multi-scale plot in Figure 5b indicate a high variability of the binning-based descriptor. Therefore large sample sizes, i.e., large windows, are required to approximate the underlying distribution to obtain clearer peaks, as indicated by the black circle. Besides, it can be noted that it suffices to categorize service times into three bins to detect the drift, even though the underlying service time distribution is bi-modal.

Consequently, the presented experimental results show that our method is able to detect drifts that require a holistic consideration of time and control flow.

VI. Conclusion and Future Work

In this paper, we proposed an EMD-based concept drift detection approach that allows for different perspectives on the process thanks to the flexibility of EMD regarding the choice of the representation and the distance measure. Furthermore, we demonstrated how a time perspective that enables service and sojourn-time aware drift detection can be implemented in this framework. The experimental evaluation showed that various types of control-flow, as well as time-dependent control-flow drifts can successfully be detected. However, the ability of the currently used trace descriptor to holistically detect drift in long-term dependencies is traded for the requirement of large windows sizes and possible additional preprocessing.

For future work, given our first results on holistic drift detection, we intend to perform a more comprehensive evaluation, in particular on real-world data, as well as to develop methods that allow to systematically find a proper preprocessing without requiring domain knowledge. Considering the trace descriptors, e.g., lower-dimensional descriptors and descriptors that incorporate additional perspectives are possible future directions to investigate. Moreover, descriptors and trace distances that are more robust to concurrency can be investigated. Furthermore, since binning the time information can result in information loss, trace distance functions that do not rely on binning need to be investigated. However, first and foremost, extensions that provide drift diagnostics, i.e., methods that reveal the actual changes of the process, are crucial to make the detected drifts actionable. For example, consider the additional peaks in Figure 5a. Currently, we can only diagnose that either control flow and sojourn times are affected or by considering only the control flow perspective conclude that only the control flow is affected. Nevertheless, in neither case we can clearly pinpoint the effect. In this context, we want to exploit the optimal values of the flow variables in the EMD-defining linear program. By construction, these values determine the EMD value and, therefore, describe the differences between the stochastic languages. Finally, the application to event streams can be investigated, which, in addition to dealing with potentially changing time information granularity, requires to adapt the trace descriptor and/or distance, e.g. by allowing partial matches, to handle incomplete traces.

References

[1] Y. Rubner, C. Tomasi, and L. J. Guibas, "A metric for distributions with applications to image databases," in *ICCV*, 1998, p. 59.

[2] S. J. J. Leemans, A. F. Syring, and W. M. P. van der Aalst, "Earth movers' stochastic conformance checking," in *BPM Forum*, 2019, pp. 127–143.

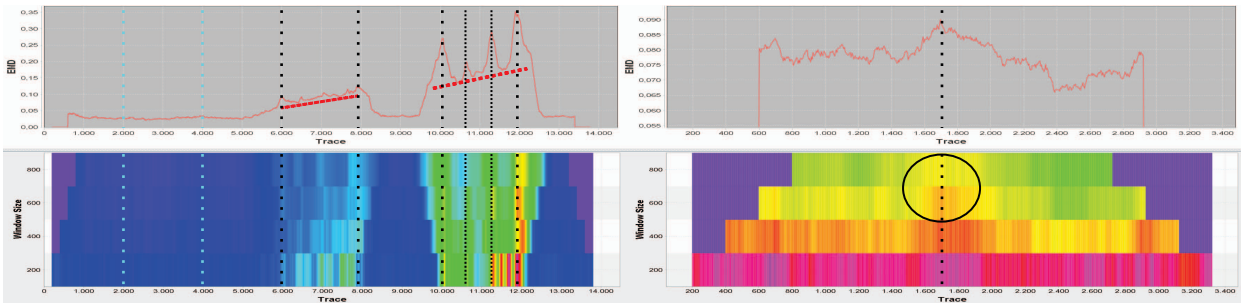
[3] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts," *Mach. Learn.*, vol. 23, Nov. 1994.

[4] A. Tsymbal, "The problem of concept drift: Definitions and related work," Tech. Rep., 2004.

[5] J. Gama, I. Žliobaitundefined, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surv.*, vol. 46, no. 4, Mar. 2014.

[6] A. Wald, "Sequential tests of statistical hypotheses," *Ann. Math. Statist.*, vol. 16, no. 2, pp. 117–186, Jun. 1945.

[7] A. Bouchachia, "Fuzzy classification in dynamic environments," *Soft Comp.*, vol. 15, no. 5, pp. 1009–1022, May 2011.



(a) Sojourn time drifts.

(b) Activity duration induced control-flow drift.

Fig. 5: Drift detection incorporating the time perspective. Window sizes 200, 400, 600, 800, selectively displaying EMD for window size 600. The stride size is 5.

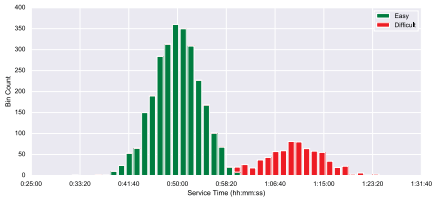


Fig. 6: Service time distribution for low claims according to the degree of difficulty.

[8] R. Klinkenberg, "Adaptive information filtering: Learning in the presence of concept drifts," in *AAAI*, 1998, pp. 33–40.

[9] D. Kifer, S. Ben-David, and J. Gehrke, "Detecting change in data streams," in *VLDB*, 2004, pp. 180–191.

[10] R. Sebastião and J. Gama, "Change detection in learning histograms from data streams," in *EPIA*, 2007, pp. 112–123.

[11] R. P. J. C. Bose, W. M. P. van der Aalst, I. Žliobaitė, and M. Pechenizkiy, "Handling concept drift in process mining," in *CAiSE*, 2011, pp. 391–405.

[12] R. P. J. C. Bose, W. M. P. van der Aalst, I. Žliobaitė, and M. Pechenizkiy, "Dealing with concept drifts in process mining," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 1, pp. 154–171, Jan. 2014.

[13] J. Martjushev, R. P. J. C. Bose, and W. M. P. van der Aalst, "Change point detection and dealing with gradual and multi-order dynamics in process mining," in *BIR*, 2015, pp. 161–178.

[14] A. Maaradji, M. Dumas, M. La Rosa, and A. Ostovar, "Fast and accurate business process drift detection," in *BPM*, 2015, pp. 406–422.

[15] W. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *TKDE*, vol. 16, no. 9, pp. 1128–1142, 2004.

[16] A. Ostovar, A. Maaradji, M. La Rosa, A. H. M. ter Hofstede, and B. F. V. van Dongen, "Detecting drift from event streams of unpredictable business processes," in *Conceptual Modeling*, 2016, pp. 330–346.

[17] A. Ostovar, A. Maaradji, M. La Rosa, and A. H. M. ter Hofstede, "Characterizing drift from event streams of business processes," in *CAiSE*, 2017, pp. 210–228.

[18] R. Accorsi and T. Stocker, "Discovering workflow changes with time-based trace clustering," in *SIMPDA*, 2012, pp. 154–168.

[19] J. Carmona and J. Cortadella, "Process mining meets abstract interpretation," in *ECML PKDD*, 2010, pp. 184–199.

[20] J. Carmona and R. Gavaldà, "Online techniques for dealing with concept drift in process mining," in *IDA*, 2012, pp. 90–102.

[21] T. Li, T. He, Z. Wang, Y. Zhang, and D. Chu, "Unraveling process evolution by handling concept drifts in process mining," in *SCC*, 2017, pp. 442–449.

[22] C. Zheng, L. Wen, and J. Wang, "Detecting process concept drifts from event logs," in *OTM*, 2017, pp. 524–542.

[23] D. Luengo and M. Sepúlveda, "Applying clustering in process mining to find different versions of a business process that changes over time," in *BPM Workshops*, 2012, pp. 153–158.

[24] B. Hompes, J. Buijs, W. van der Aalst, P. Dixit, and J. Buurman, "Detecting change in processes using comparative trace clustering," in *SIMPDA*, 2015, pp. 95–108.

[25] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *SODA*, 2007, pp. 1027–1035.

[26] D. Liu and J. Yu, "Otsu method and k-means," in *HIS*, vol. 1, 2009, pp. 344–349.

[27] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *JACM*, vol. 21, no. 1, pp. 168–173, Jan. 1974.

[28] A. Marzal and E. Vidal, "Computation of normalized edit distance and applications," *TPAMI*, vol. 15, no. 9, pp. 926–932, 1993.

[29] T. Dasu, S. Krishnan, S. Venkatasubramanian, and K. Yi, "An information-theoretic approach to detecting changes in multi-dimensional data streams," in *Symp. Int. Comp. Scie. Stat.*, 2006.

[30] K. Paparrizos, "A non improving simplex algorithm for transportation problems," *RAIRO-OPER RES*, vol. 30, no. 1, pp. 1–15, 1996.

[31] H. Achatz, P. Kleinschmidt, and K. Paparrizos, "A dual forest algorithm for the assignment problem," in *Applied Geometry and Discrete Mathematics: The Victor Klee Festschrift*, vol. 4, 1991, pp. 1–10.

Collaborative and Interactive Detection and Repair of Activity Labels in Process Event Logs

Sareh Sadeghianasl, Arthur H.M ter Hofstede, Suriadi Suriadi, Selen Turkay
Queensland University of Technology, Brisbane, Australia
{s.sadeghianasl, a.terhofstede, s.suriadi, selen.turkay}@qut.edu.au

***Abstract*—Process mining uses computational techniques for process-oriented data analysis. The use of poor quality input data will lead to unreliable analysis outcomes (garbage in - garbage out), as it does for other types of data analysis. Among the key inputs to process mining analyses are activity labels in event logs which represent tasks that have been performed. Activity labels are not immune from data quality issues. Fixing them is an important but challenging endeavour, which may require domain knowledge and can be computationally expensive. In this paper we propose to tackle this challenge from a novel angle by using a gamified crowdsourcing approach to the detection and repair of problematic activity labels, namely those with identical semantics but different syntax. Evaluation of the prototype with users and a real-life log showed promising results in terms of quality improvements achieved.**

***Index Terms*—Process mining, data quality, event log, activity label, gamification, crowdsourcing.**

I. Introduction

Process mining is a well-established research area that concerns discovering, controlling, and improving processes using data recorded in event logs [1]. Unfortunately, event logs in practice suffer from numerous data quality issues, e.g. imprecise timestamps [2], infrequent behavior [3], and duplicate events [4]. Using an event log of poor quality as a starting point for process mining would constitute a case of the well-known adage "garbage in - garbage out". One type of quality problem in event logs relates to *activity labels* which reflect the tasks that were performed, e.g. "create purchase order". It is important to obtain the correct labels in order to avoid treating tasks that are identical as different or different tasks as identical. The former challenge concerns finding *synonymous* labels with different syntax, *distorted* labels with minor syntactic differences, and *polluted* labels with the same syntax structure (using the terminology of the event log imperfection patterns [5]). The latter challenge amounts to the problem of identifying *homonymous* labels [5]. The focus of our paper is on the former challenge, i.e. correcting and/or standardising labels that exhibit a different syntax but have the same underlying semantics. We will refer to these types of labels as *imperfect* in the remainder of this paper. Also, we will use the terms *activity* and *label* interchangeably to refer to activity labels. The problem of imperfect labels is acute and tackling this problem requires a novel approach in order to progress beyond the state of the art. In particular, we believe the involvement of domain experts is essential to achieve a breakthrough. And while they have been leveraged before, we do not only focus on label detection but also repair. In addition, we believe that domain experts need to be involved in an iterative fashion so that learning can be cumulative. Hence it is essential that our approach (C1) converges towards an improved log, (C2) supports input from multiple domain experts, and (C3) engages them in order to encourage their participation.

To achieve criterion C2, we will use a form of crowdsourcing. Crowdsourcing in general is defined as "the act of taking a job traditionally performed by a designated agent (usually an employee) and outsourcing it to an undefined, generally large group of people in the form of an open call" [6]. It can provide a diversity of skills and knowledge for detecting and repairing imperfect activities in a more efficient way, while most computational approaches underestimate the complexity of such a task [7]. In our approach, we further determine the level of domain expertise using a number of strategies.

Gamification techniques will be used to satisfy criterion C3. Gamification is defined as "the use of video game elements in non-gaming systems to improve user experience (UX) and user engagement" [8] and is growing in popularity in various domains, e.g. healthcare [9], business [10], and information systems [11]. While gamification has been used in crowdsourcing successfully (see [12] for a review), there is limited attention to gamification in process mining, let alone in improving the quality of event logs. There are numerous game elements (e.g. narrative, choice) used to enhance user engagement with gamified systems and to improve their motivation to return to the system [13]. This study uses common gamification elements, e.g. points, levels, and badges [14] to inspire feelings of competence in people and motivate them to perform tasks more efficiently, and provide better advice for detecting and repairing imperfect labels.

This paper is structured as follows. Section II describes related work. In Section III we discuss the design framework of our approach. In Section IV we describe a prototype of our gamified system. Section V presents our evaluation and Section VI summarises our approach and suggests future work.

II. Related Work

The study of event log quality was initially discussed in the Process Mining Manifesto [15] where a 1 to 5 star spectrum was proposed for rating the quality of an event log. In such a framework, process mining analysis can be applied on logs rated as 3, 4 or 5-star, but not on 1 and 2-star logs. The

978-1-7281-9832-3/20/$31.00 ©2020 IEEE
DOI 10.1109/ICPM49681.2020.00017

works in regards to event log quality can be divided into two main categories: those that propose frameworks for event log quality dimensions (e.g. [5], [16]–[19]), and those that propose methods for improving event log quality (e.g. [2]–[4], [20], [21]). In this section we review the second category as we are pursuing the same goal in this paper.

Among the approaches that have been proposed to improve the quality of event logs, e.g. [2]–[4], [20], [21], some address noise and outliers [22], some are concerned with event timestamp and ordering quality issues [2], and some others focus on activity label quality issues [4], [20], [21], [23], which is the general topic area of this paper. Some of theses approaches [20], [23], [24] have looked at activity label quality issues at the process model level, among which van der Aa et al. [23] and Koschmider et al. [20] discuss labels that stem from the same model and Pittke et al. [24] covers labels that stem from a pool of process models from different domains [24]. In the field of process model matching and similarity [25], label similarity measures have been introduced and may be useful in detecting imperfect activity labels which have different syntax but the same semantics. Some other approaches have addressed activity label quality issues at the event log level [4], [21], however they mainly focus on detecting and repairing activity labels with the same syntax but different semantics, i.e. *homonymous* labels [5]. Our approach, instead, focuses on activities with the same semantics but different syntax, i.e. *synonymous*, *distorted*, and *polluted* labels (as defined in [5]) of a single process at the event log level.

The use of domain knowledge in improving activity quality has been shown to be effective [7], [26]–[28]. This domain knowledge can be acquired from an external data set, e.g. an ontology or a dictionary [20], or it may be provided by a human (either a limited number of individuals [26], [27] or an unlimited crowd [7], [28]). Expert feedback has been used to suggest activity matching [26] or correct automatically generated matching between activities of two process models [27]. Some approaches [7], [28] use crowdsourcing to identify activity matches and argue that such a matching is not as straightforward as most automatic methods assume. These approaches detect activity matches while, in our approach, we use crowd-sourced domain knowledge to *detect* and also *repair* activities with the same meaning. In addition, we use gamification techniques to improve the engagement of the crowd in completing the required tasks.

III. Approach Design

Improving the quality of activity labels in an event log leads to more reliable process mining outcomes [4], [21]. Figure 1 illustrates our approach, which takes an event log as input and produces a repaired event log as output which is of higher quality. This log can be improved even further if it serves as the input for another round of data cleaning through the use of the gamified system (Criterion C1). As depicted in Figure 1, our approach collects knowledge of multiple domain experts (a form of crowdsourcing) to detect and repair imperfect activity labels in a collaborative and interactive manner (Criterion C2). Crowdsourcing enables us to work with knowledgeable people in a given domain without being restricted by their physical location. As part of our approach we further determine the level of domain expertise by asking people to declare how familiar they are with a particular process and also by asking them some baseline assessment questions.

In order to motivate domain experts towards providing advice for detecting and repairing imperfect activities in the log, we make use of gamification techniques (Criterion C3). Based on the *Octalysis* gamification design framework [29], our approach supports a set of motivational drives as follows:

- *Development & Accomplishment*: This is the internal drive of most people for overcoming challenges, gaining knowledge, making progress, and achieving goals and mastery. In our approach, we have designed collectable points, levels, badges and progress bars to set goals for the players and inspire their feelings of achievement. We also regularly narrate interesting facts regarding data quality to support acquiring knowledge.
- *Epic Meaning & Calling*: This motivational drive is in play when a person assumes they are doing something bigger than themselves, e.g. saving the world. In our approach, we regularly narrate real disasters that happened due to poor data quality and caused many lost lives or led to high costs. This gives domain experts some context about why they should engage in the game and enables them to imagine how they can potentially prevent similar disasters in the future.
- *Social Influence & Relatedness*: This drive motivates people to follow a social norm, i.e. what everyone else is doing. In our approach, we motivate players by providing a 'help' option that shows the number and the views of people who have tried the game challenges so far.

IV. Gamified System Prototype

As depicted in Figure 1, the gamified system consists of three phases: (1) the *pre-game phase*, where we generate a question bank from an input log, (2) the *game phase*, where the players interact with the game to suggest quality improvements, and (3) the *post-game phase*, where we repair the log based on players' answers. The following sections describe each of these phases in detail.

Let us first formalize the notion of an *event log*. Let $\mathcal{A}$ be a set of activities. A *trace* σ is a *sequence* of activities, i.e. $\sigma \in \mathcal{A}^*$. An event log L is a multi-set of traces over $\mathcal{A}$, i.e. $L = \{(\sigma, n) \mid \sigma \in \mathcal{A}^*, n \in \mathbb{N}\}$ where n is the number of times the trace σ is repeated in log L.

A. Pre-game

In the pre-game phase, we generate a set of multi-choice questions, each asking players to select all *options*, in the form of activities, that are synonyms to a given *main activity*. The *main activity* and the *options* are chosen from *imperfect* labels in the log, as they are the candidates for repair actions.

In order to detect imperfect labels, we define a notion called *uniqueness factor*. A uniqueness factor represents the extent

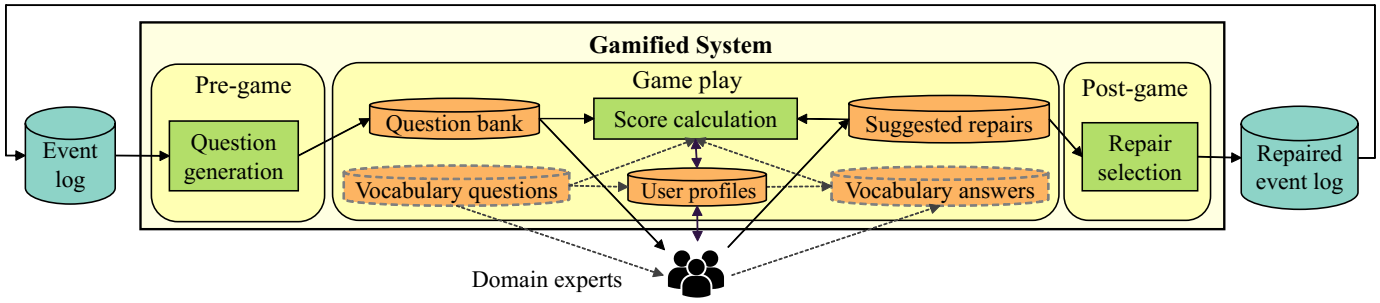


Fig. 1. A gamified system to detect and repair imperfect activity labels

to which an activity has unique semantics compared to other activities in the log. The uniqueness factor $uf : \mathcal{A} \rightarrow [0, +\infty)$ for any activity $a \in \mathcal{A}$ occurring in log L, is defined as:

$$uf(a) = 2*(1-S_{max}(a)) + \frac{1}{|\mathcal{A}|-1} \Sigma_{x \in \mathcal{A}-\{a\}} |S(a,x) - S_{max}(a)| \quad (1)$$

where for any activities $a, x \in \mathcal{A}$, $S(a,x)$ is the normalized weighted average similarity between a and x, defined in our previous work [30], which takes into account context similarities of activities from four dimensions: control flow, resource, data, and time; and $S_{max}(a) = \max_{x \in \mathcal{A} \setminus \{a\}} S(a,x)$, is the maximum similarity between a and any other activities in the log. $uf(a)$ depends on $S_{max}(a)$ and the average difference between $S_{max}(a)$ and $S(a,x)$. Lower values of $S_{max}(a)$, i.e. a has a unique semantics in the log, lead to lower values of $uf(a)$. Conversely, higher values of $S_{max}(a)$ lead to higher values of $uf(a)$ and this effect becomes greater the closer the similarity between a and other activities in the log are to $S_{max}(a)$. We have introduced a multiplication factor of two to make $uf(a)$ more sensitive to the value of $S_{max}(a)$ rather than to the sum of the absolute differences between $S(a,x)$ and $S_{max}(a)$, because activities that are similar to even one other activity in the log are candidates of imperfect labels and this should be reflected in the value of their uf.

Algorithm 1 presents our technique for generating questions using an event log. Each question focuses on one group of candidate imperfect activities with the same meaning. Initially, we mark all activities as not visited $\mathcal{A}_{nv}$ and find the activity $mainAct \in \mathcal{A}_{nv}$ that has the minimum uniqueness factor among them (the function called at Line 3). Then we sort other activities in $\mathcal{A}_{nv}$ descending by their similarity to $mainAct$ (the function called at Line 4) and pick the first $\lceil m/2 \rceil$[1] activities as similar options if their context similarity with $mainAct$ is higher than a threshold θ (Lines 6–7). Then, we fill out the rest of the options with activities that are not similar to $mainAct$ selected from the end of the list (Line 8). Finally we mark $mainAct$ and its similar options as visited because we do not want to see them in other questions and then we proceed to generating the next question until the desired number of questions n is reached[2]. The time complexity of Algorithm 1 is $O(nk \log k + nm)$ where k, n, and m are the number of activities, questions and options per question respectively.

Algorithm 1: Generate Game Questions

Input: Activity universe $\mathcal{A}$, number of questions n, number of options m, threshold θ
Output: Question bank $\mathcal{Q}$

1 $\mathcal{A}_{nv} \leftarrow \mathcal{A}, \mathcal{Q} \leftarrow \{\}$;
2 **while** $|\mathcal{Q}| < n$ **do**
3 $mainAct \leftarrow$ findMinUFActivity($\mathcal{A}_{nv}$);
4 $\mathcal{A}_{sorted} \leftarrow$ sortDescBySimTo($\mathcal{A}_{nv}, mainAct$);
5 $O_{sim} \leftarrow \{\}, O_{diff} \leftarrow \{\}$;
6 **for** $i \leftarrow 0$ **to** $\lceil m/2 \rceil$ **do**
7 **if** $S(\mathcal{A}_{sorted}[i], mainAct) \geqslant \theta$ **then** $O_{sim} \leftarrow O_{sim} \cup \{\mathcal{A}_{sorted}\ [i]\}$;
8 **for** $j \leftarrow 0$ **to** $m - |O_{sim}|$ **do**
9 $O_{diff} \leftarrow O_{diff} \cup \{\mathcal{A}_{sorted}\ [|\mathcal{A}_{sorted}|\ - j]\}$
10 $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(mainAct, O_{sim} \cup O_{diff})\}$;
11 $\mathcal{A}_{nv} \leftarrow (\mathcal{A}_{nv} - \{mainAct\}) - O_{sim}$;
12 **return** $\mathcal{Q}$

B. Game play

The gamified system is designed around a number of player goals. Each goal corresponds to achieving a certain level, represented by a badge received for collecting a set number of points. We have designed two point schemes to measure different achievements: *stars* scheme which measures a player's experience (XP) with the game to encourage players' participation in the game [31], and *diamonds* which measures a player's knowledge of the domain so that the answers provided can be weighted properly. Accordingly, two sets of levels are defined: XP levels and knowledge levels. The players can earn both XP and knowledge points by overcoming some *challenges*. A *challenge* consists of three parts:

- A *Baseline assessment* part, which is a yes/no question asking whether two terms are synonyms. These questions are acquired from the vocabulary question bank

[1]This limitation of the number of similar options is to preserve balance.

[2]We acknowledge the fact that n and m need to be bounded on implementation, but conceptually, we do not spell it out for simplicity.

(Figure 1), which includes a set of true synonyms or different terms in a domain[3]. These questions are used to assess a player's level of familiarity with the domain from which the event log stems. This part is the only part for which we know the ground truth and if the player answers this part correctly, they will earn more knowledge score (diamonds) and as a result in the post-game phase we will value their answers more (i.e., higher weightings). We are inspired by the idea of using a control word in the reCAPTHA system [32] for determining correct words scanned from old print materials.

- A *Detection* part, which is a multi-choice question asking the player to select all options that are synonyms of a given *main activity*. These questions are derived from the question bank (Figure 1) and help us to identify imperfect activity candidates in the event log.
- A *Repair* part, which is a multi-choice question with an optional input field asking a player to suggest a replacement for the synonyms he has chosen in the Detection part. The player can either select one of the options or suggest a new one through the input field.

Players receive immediate feedback on the number of stars and diamonds earned after each challenge. The number of earned stars $s \in \mathbb{N}$ is computed as $s = \lfloor \alpha * p \rfloor$, where $p \in \{1, 2, 3\}$ is the number of completed parts of the challenge by the player and $\alpha > 0$ is a constant factor. The more parts of the challenge completed the more stars earned to represent the player's experience with the game.

Players earn a number of diamonds $d \in \mathbb{N}$, measuring their domain knowledge, in two parts: the Baseline assessment part $d_b \in \mathbb{N}$ and the Repair part $d_r \in \mathbb{N}$[4], i.e. $d = d_b + d_r$. The Baseline assessment score is calculated as $d_b = \beta * v$ where $\beta > 0$ is a constant factor and $v = 1$ if the Baseline assessment question is answered correctly and $v = 0$ otherwise.

The Repair score measures the suitability of a suggested repair as a string and is calculated as $d_r = d_l + d_s$ where $d_l \in \mathbb{N}$ is the length score and $d_s \in \mathbb{N}$ is the spacial characters and numbers score. The length score follows a bell-shaped function[5] and is calculated as $d_l = \gamma * f(x; a, b, c) = \gamma * \frac{1}{1+|\frac{x-c}{a}|^{2b}}$, where $x \in \mathbb{N}$ is the length of the repair, $\gamma > 0$ is a constant factor, and $a, b, c > 0$ are the function parameters specifying width, slope and center of the bell curve respectively. The special characters and numbers score is computed as $d_s = \eta * sc$, where $\eta > 0$ is a constant factor and $sc = 0$ if some special characters and numbers exist in the suggested repair and $sc = 1$ otherwise.

C. Post-game

In the post-game phase, we repair the log using collected answers from players. The idea is to select answers of high weight, where the weight of an answer is determined by a player's knowledge level, associated with the number of earned diamonds in the game, and the player's original self-declared expertise level in the domain. This method is referred to as *vote weighting* in crowdsourcing [33].

Let $\Lambda \subset \mathcal{Q} \times \Phi \times \mathcal{R}$ be the set of all answers given by players to game challenges, where $\mathcal{Q} \subseteq \mathcal{A} \times (2^{\mathcal{A}} \backslash \{\varnothing\})$ is the question bank generated in the pre-game phase (Section IV-A), each question consisting of a main activity and a set of imperfect activities as options, $\Phi \subseteq 2^{\mathcal{A}}$ is the set of all detected imperfect activities (in the Detection part of each challenge), and $\mathcal{R} \subseteq \Sigma^*$ is the set of all repairs suggested by players (in the Repair part of each challenge) where Σ is the set of all characters. An answer $\lambda \in \Lambda$ is defined as $\lambda = (\lambda_q, \lambda_{\mathcal{A}}, \lambda_r)$ where $\lambda_q \in \mathcal{Q}$, $\lambda_{\mathcal{A}} \subseteq \Phi$, and $\lambda_r \in \mathcal{R}$ refer to the question, the set of detected imperfect activities, and the repair associated with λ. The weighting function $W : \Lambda \rightarrow [0, +\infty)$, assigns a weight to an answer $\lambda \in \Lambda$ based on the knowledge level $k \in \mathbb{N}$ and the original expertise level $e \in \mathbb{N}$ of a player and is defined as $W(\lambda) = \tau_k * k + \tau_e * e$ where $\tau_k, \tau_e \geqslant 0$ are the corresponding coefficients.

For any activity $a \in \mathcal{A}$ and any repair $r \in \mathcal{R}$, the repair weight function $\Psi : \mathcal{A} \times \mathcal{R} \rightarrow [0, +\infty) \cup \perp$ is defined as $\Psi(a, r) = \Sigma_{\lambda \in \Lambda, a \in \lambda_{\mathcal{A}}, \lambda_r = r} W(\lambda)$ which is the sum of weights of the answers that have suggested r as a repair and have a in their detection set. Also, the crowd-based similarity $\Delta : \mathcal{A} \times \mathcal{A} \rightarrow \mathbb{R}$ between any two activities $a, b \in \mathcal{A}$ is defined as:

$$\Delta(a, b) = \Sigma_{\lambda \in \Lambda : \{a,b\} \subseteq \lambda_{\mathcal{A}}} W(\lambda) - \Sigma_{\lambda \in \Lambda : ((a \in \lambda_{\mathcal{A}}, b \in Opt(\lambda_q) \backslash \lambda_{\mathcal{A}}) \vee (b \in \lambda_{\mathcal{A}}, a \in Opt(\lambda_q) \backslash \lambda_{\mathcal{A}}))} W(\lambda)$$

where $Opt : \mathcal{Q} \rightarrow 2^{\mathcal{A}}$ is a function that retrieves the set of activity options from a question. If a and b are detected as similar in an answer λ we increase their crowd-based similarity by the weight of λ. Otherwise, if only one of a and b is in the detection set while the other one has been provided as an option in the question, this means that they are detected as different by a player and we decrease their crowd-based similarity by the weight of λ.

Algorithm 2 presents our method for selecting the final activity repairs to be applied on the log. We repeatedly find the overall maximum weighted repair r_m for any activity a_m[6] and if the weight is more than a similarity threshold θ_s (Line 2), we add the pair to the selected repairs set Π and also select the same repair r_m for any activity b which is very similar to a ($\geqslant \theta_s$) (Line 3). This is to guarantee that all activities that are detected as similar are repaired to a single label. Also in Lines 4 and 5 we remove a set of repair pairs Γ from the domain of the repair weight function Ψ to ensure that they will not be chosen in the next iterations[7]. More specifically, for any activity that has a selected repair, i.e. a_m and any of its similar activities, we remove all other possible repair pairs and for activities that are very different form a_m ($\leqslant \theta_d$)

[3]The vocabulary question bank can be generated by a domain expert or by using a domain-specific resource. We acknowledge that this might affect the applicability of the approach on specific domains.

[4]We ignore the Detection part due to the lack of the relevant ground truth.

[5]We penalize too short and too long repairs because based on our observations in real-life logs, too short labels often do not describe the relevant semantics precisely and too long labels often include unnecessary information.

[6]The function $argmax\ \Psi$ gives the argument that maximizes Ψ.

[7]$\lhd$ represents the domain restriction operator in the formal language Z.

we remove r_m from their possible repairs to make sure that different activities are not repaired into the same label. The output of the algorithm is the set of selected repairs Π which can be applied to the input log resulting in a repaired log. The time complexity of Algorithm 2 is $O(k^2r^2)$ where k and r are the number of activities and suggested repairs respectively.

Algorithm 2: Select Repairs

Input: Crowd-based similarity function Δ, repair weight function Ψ, similarity threshold θ_s, difference threshold θ_d

Output: Selected repairs Π

Precondition: $dom\ \Psi \neq \varnothing$

$\Pi \leftarrow \{\}$;

while $(a_m, r_m) \leftarrow argmax\ \Psi, \Psi(a_m, r_m) \geqslant \theta_s$ **do**

  $\Pi \leftarrow \Pi \cup \{(a_m, r_m)\}$
    $\cup\{(b, r_m) \in dom\ \Psi | \Delta(a_m, b) \geqslant \theta_s\}$;

  $\Gamma \leftarrow \{(a_m, r) | r \in \mathcal{R}\}$
    $\cup\{(x, r) \in dom\ \Psi | \Delta(x, a_m) \geqslant \theta_s\}$
    $\cup\{(c, r_m) \in dom\ \Psi | \Delta(c, a_m) \leqslant \theta_d\}$;

  $\Psi \leftarrow (dom\ \Psi \backslash \Gamma) \lhd \Psi$;

return Π

V. Implementation and Evaluation

We have implemented our gamified system, called *The Quality Guardian* to be used to evaluate our approach with a public log and a number of participants. This gamified approach can also be used in an organizational setting with a particular data set.

A. Data

We used the BPIC 2019 event log [34] for our evaluation. This log contains events related to a purchase order process of a painting company. The log was a pre-processed version of a real-life log and was free from imperfect labels, however, we purposely injected some imperfect labels to test our approach. We selected a subset of distinct activity labels (8 out of a total of 42) and renamed a portion of their instances with new labels containing similar semantics based on patterns that we have encountered in real-life logs (i.e. *synonymous*, *distorted*, and *polluted*) [5][8]. The proportion of the label instances, for each of these 8 activities, that were renamed is in the range of 20% to 50% because this is roughly the proportion of label instances that were observed to be imperfect in real-life logs. From such an injection, we obtained an event log with 8 groups of imperfect labels[9]. Each group consists of two or more distinct labels with the same semantics, e.g. group 3 consists of 5 labels: "Clear Invoice", "Clr Invoice", "Remove Invoice", "Eliminate Inv", and "Clear Inv".

[8]We acknowledge that artificially injecting imperfect labels might not fully capture real-world issues. However, using a real-life log with actual imperfect labels but without access to domain experts in an experimental setting is challenging due to *the lack of ground truth*. Conducting such an evaluation is left as future work.

[9]Available at https://event-log-quality.s3.amazonaws.com/icpm/log.xes.

After injecting imperfect labels, we applied Algorithm 1 to generate 8 questions, each for one group of imperfect activities and each with 9 options, i.e. $n = 8, m = 9$. We chose 9 options because similar options per question in this log add up to at most 4 and we added 5 different options to preserve balance. We used a similarity threshold (θ) of 0.65 so that more activities can be included in the options. For computing the weighted average similarity between activities as per Equation 1, the weights we used for the context dimensions are 5, 5, 1, and 1, for the control-flow, resource, time, and data dimensions respectively [30]. The weights used are in line with those used in our previous work [30] where we assigned the control-flow and resource dimensions higher weights because we found them more informative in measuring the similarity between activities. The execution of Algorithm 1 took about 50 seconds on a Windows 64-bit operating system with an Intel Core i7 7600 processor and a 16GB RAM.

B. The Game

Players are initially introduced to a game character called *Sam* (Figure 2(a)) who is a data analyst. Players are requested to help *Sam* resolve some of the imperfect labels that he has seen in his data set, i.e. an event log. To trigger a player's engagement [31], we let them experience the game before signing up. Sam asks the players a few sample questions, each relating to the three main parts of the game: *Baseline assessment*, *Detection*, and *Repair* (Section IV-B).

Once the players are familiar with the game, they are asked to sign up to help Sam with his data set. During sign-up, players are requested to declare their expertise level w.r.t the domain of the event log: *nothing* (i.e. no domain knowledge whatsoever), *a little*, *a fair bit*, and *a lot*. This level cannot be changed or increased during the game. The players are granted 10 stars and an XP badge *Good Starter* as soon as they join the game (Figure 2(b)) as a reward to motivate them to further interact with the game [31]. Next, the players see a demo of the game (skippable) before proceeding to the actual game.

Figures 2(c), (d), and (e) show the screenshots of our implementation of the three main parts of the game: Baseline Assessment, Detection, and Repair (Section IV-B). For the Baseline assessment, we created eight yes/no questions using common synonyms for typical terms used in purchasing orders, e.g. "refund" and "money back". To assist and motivate players, we provide players with 'help' options in the forms of statistical data, such as the *measured context similarities* [30] between the main activity and other labels in the options, the *absolute and relative frequencies* of each label in the log, the *popularity* of each activity which is the number of previous players who chose a particular option for a given question, and the *string suitability metric of suggested repairs* (only for the Repair part) as measured by the repair score d_r in Section IV-B. Note, the label popularity measure that a player sees will not be the same as another player, depending on how many players have completed the game prior.

A perfect answer for each challenge is rewarded 10 stars and 10 diamonds. The designated XP badges are *Good Starter*,

Newbie Contributor, and *Phenom Contributor*, each requiring 10, 40, and 80 stars respectively. Similarly, the available Knowledge Badges are *Problem Solver*, *Expert*, and *Master*, each requiring 20, 45, and 80 diamonds respectively. These thresholds are chosen based on the number of challenges used in the evaluation, i.e. 8, and considering the idea that achieving higher levels would be naturally more difficult than the lower ones [31]. Players can view their game status through their profile page (Figure 2(f)).

C. Evaluation Setup

The Quality Guardian game was implemented as a Java web applications hosted by a Tomcat server[10]. It was available online for 3 weeks and was advertised using social media, e.g. Linkedin. We did not limit the expertise field of players to the specific domain of the event log, i.e. purchasing orders, because the game allows for different levels of domain expertise and because our algorithm prefers the responses by domain experts in selecting the final labels to repair the log.

We have used the formulas introduced in Section IV-B to score players during the game. Once we captured all responses from players, we applied Algorithm 2 to select the best suggested repairs. We selected the values of the similarity and difference thresholds θ_s and θ_d such that the similarity or the difference of two activities is approved only if they have a weighted vote of at least half of the maximum. The parameters are detailed in the appendix[11]. The execution of Algorithm 2 took about 43 seconds on the same system used for running Algorithm 1.

We measured the effectiveness of our approach in a number of ways. A starting point is to see how many similar labels have been reduced in the repaired log. Secondly, we evaluate whether our approach aggregates correct sets of labels by looking at the number of distinct activity labels in each group of imperfect labels. Finally, to measure the overall effectiveness of our approach, we used the adapted version of the *Levenshtein distance* [35] (used by Conforti et al. [2]) to the concept of *cases* whereby, instead of counting addition, deletion, or replacement of a *character*, we count addition, deletion, or replacement of an *event*.

At the end of the game, to assess Criterion C3, we asked players to complete an online survey with 5 point Likert scale answers (strongly agree-strongly disagree) and describe how they felt about the game. Specifically, they were asked 8 questions each about one of the desirable features of a game as per the GameFlow evaluation framework [36], including (1) the clarity of the goals, (2) the usefulness of the feedback they received, (3) the ease of use, (4) the usefulness of knowing the crowd views, (5) the ability to control the game actions and interface, (6) the engagement, (7) the knowledge development, and (8) the required concentration. We use the Cronbach's alpha coefficient which is a measure of a multi-user Likert scale survey responses' reliability [37].

[10]Available at https://event-log-quality.s3.amazonaws.com/icpm/codes.zip.

[11]Available at https://event-log-quality.s3.amazonaws.com/icpm/Pars.pdf. Although one can set other parameters, all players are scored with the same parameters, thus changing them would not affect the selection of repairs.

D. Results

A total of 27 players participated in the game, out of which 6 opt out and we are left with 21 players who completed all questions and the survey. In the original log, there were 42 distinct activities which was increased to 67 after injection. After the evaluation, the repaired log contains 46 labels where 6 are injected labels and 40 are from the original log. This means that 2 of the original labels are removed from the log as part of the repair actions. This can happen because the players preferred an injected label more than the original one with the same meaning. We do not expect this to be a problem in real setting where all options will be chosen from real imperfect activities and choosing any of them as a repair would be acceptable.

Figure 3(a) shows the number of distinct labels in each of the eight imperfect groups of labels in the original, injected, and the repaired logs. As depicted in Figure 3, there was originally one label in each group which was increased to 3–5 labels per group as a result of injection. Through our repair approach, we have been able to reduce the number of distinct labels in each group to one in most groups except for three groups, i.e. G4, G5, and G7. This chart shows that our approach managed to remove some ambiguity in activity labels (as required by criterion C1 described in Section I) by grouping similar labels into one. The remaining multiple labels in each group may be addressed by decreasing the similarity threshold θ_s or by increasing the number of players which increases the weights of responses allowing them to pass θ_s.

Table I shows the average distance between the original log (ground truth written as GT), and the injected log as well as the repaired log along with other metrics. We can see that the repaired log *indeed* is closer, in terms of Levenshtein distance, to the ground truth compared to the injected log. In other words, our approach which exploits gamification and crowdsourcing methods, has delivered some success in repairing an imperfect log closer to the ground truth.

TABLE I
FEATURES AND MEASURES OF THE INJECTED AND REPAIRED LOGS

Log	#Acts	#Events	#New acts	#Aff events	Dist to GT
Injected	67	1595923	23	321070	0.2012
Repaired	46	1595923	6	58030	0.0364

Figure 3(b) summarises the results of the survey for each of the eight features (a score of 1, 5, and 3 refer to 'strongly disagree', 'strongly agree', and 'neutral' respectively). The overall Cronbach's alpha coefficient of 0.7341 in this survey shows that it is of acceptable reliability. The maximum achieved score is 4.62 for Q3 (the ease of use) and the minimum score is 2.66 for Q4 (the usefulness of knowing the crowd views). We have observed that some players did not know how to use crowd views because they have skipped the initial demo which teaches them how to do so and this explains the reason for this score. The score for other questions

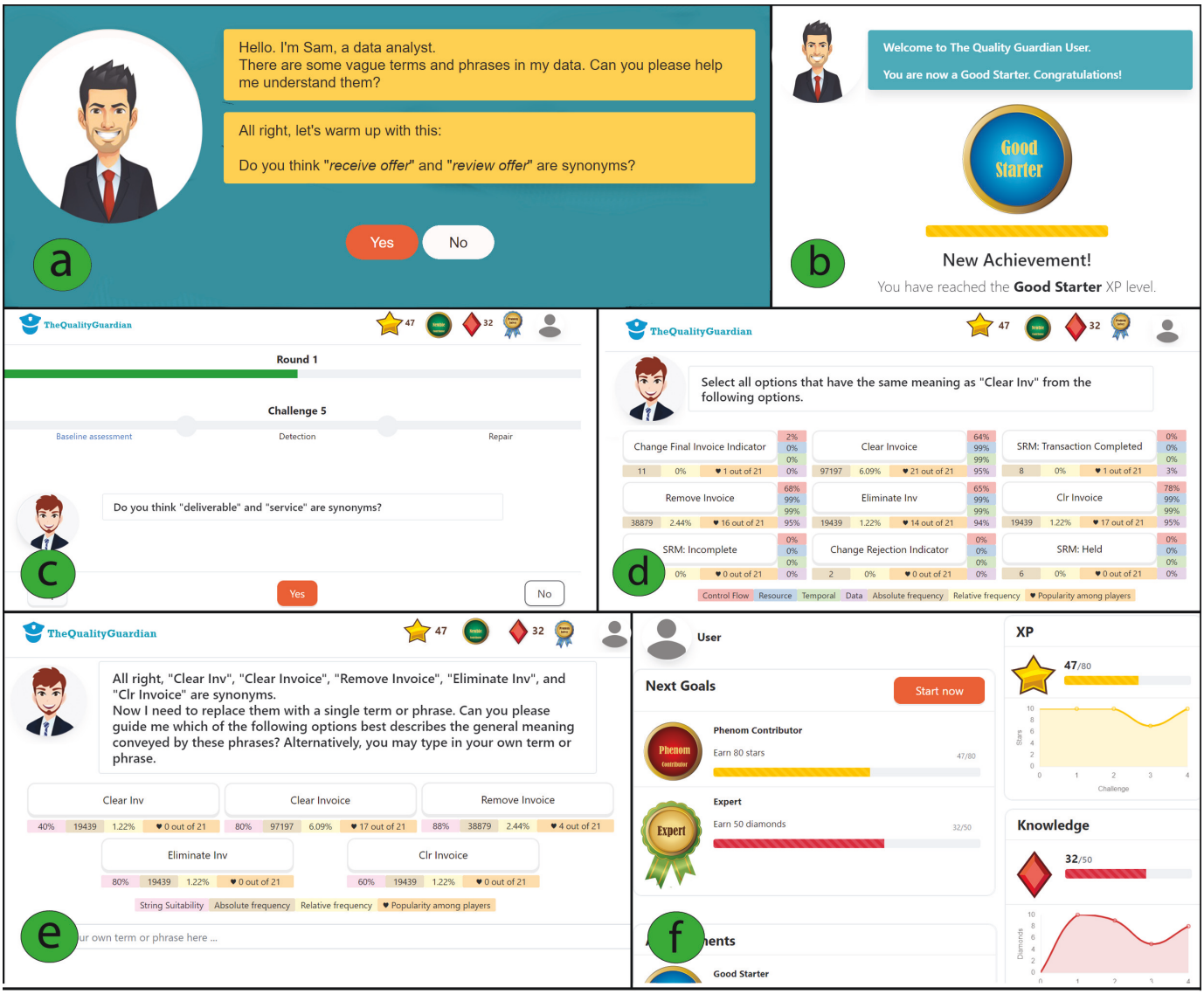


Fig. 2. Screenshots of the game: (a) main page, (b) badge granted on signing up, (c) baseline assessment, (d) detection, (e) repair, and (f) profile page.

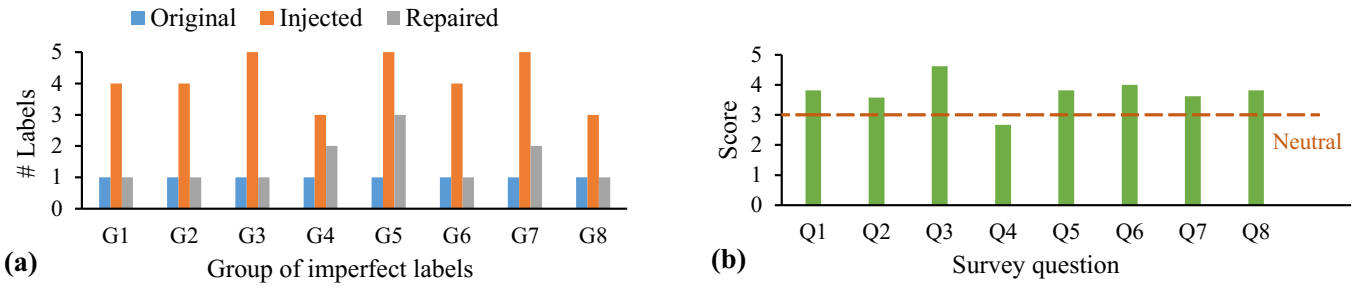


Fig. 3. (a) The number of distinct labels in each imperfect group in the original, injected, and repaired logs, (b) Summary of survey responses

is between 3.5–4, e.g. 4 for Q6 which is the engagement of the game, and this confirms that gamification is a promising technique to improve user experience (Criterion C3).

VI. Conclusion

We have proposed an approach for detecting and repairing imperfect activity labels. Our approach uses human knowledge through crowdsourcing and gamification techniques to repair event logs with imperfect labels. We have implemented a prototype of our gamified system and evaluated our approach by engaging a number of players in the game to repair a real-life event log with injected imperfect labels. The results suggest that the approach is effective in improving the quality of activity labels in event logs. The survey responses suggest that we have been successful in developing a gamified system for our purpose as most players have found it quite engaging. Some of the limitations to our approach include relying on the existence of multiple domain experts and not evaluating the approach with real imperfect labels. Further assessment could be done using additional logs and the use of process mining techniques to compare before-repair and after-repair models. Some other avenues for future work include the incorporation of additional motivational drives [29] in the gamified system and addressing other types of quality issues in event logs through the deployment of similar techniques.

VII. Acknowledgement

We acknowledge Moe Wynn's contributions through her involvement in discussing this new idea and her reviewing of the paper.

References

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

[2] R. Conforti, M. La Rosa, A. H. M. ter Hofstede, and A. Augusto, "Automatic Repair of Same-Timestamp Errors in Business Process Event Logs," in *BPM Conference*, 2020 (in press).

[3] R. Conforti, M. L. Rosa, and A. H. M. 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.

[4] X. Lu, D. Fahland, F. J. van den Biggelaar, and W. M. P. van der Aalst, "Handling Duplicated Tasks in Process Discovery by Refining Event Labels," in *BPM Conference*, ser. LNCS, vol. 9850. Springer, 2016, pp. 90–107.

[5] S. Suriadi, R. Andrews, A. H. M. 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.

[6] J. Howe, "The rise of crowdsourcing," *Wired magazine*, vol. 14, no. 6, pp. 1–4, 2006.

[7] C. Rodríguez, C. Klinkmüller, I. Weber *et al.*, "Activity matching with human intelligence," in *BPM Forum*, ser. LNBIP, vol. 260. Springer, 2016, pp. 124–140.

[8] S. Deterding, M. Sicart, L. E. Nacke *et al.*, "Gamification: Using Game-Design Elements in Non-Gaming Contexts," in *CHI Extended Abstracts on Human Factors in Computing Systems*. ACM, 2011, pp. 2425–2428.

[9] A. S. Miller, J. A. Cafazzo, and E. Seto, "A game plan: Gamification design principles in mHealth applications for chronic disease management," *Health Informatics Journal*, vol. 22, no. 2, pp. 184–193, 2016.

[10] J. Kumar, "Gamification at Work: Designing Engaging Business Software," in *DUXU*, ser. LNCS, vol. 8013. Springer, 2013, pp. 528–537.

[11] C. Schlagenhaufer and M. Amberg, "A Descriptive Literature Review and Classification Framework for Gamification in Information Systems," in *ECIS*, no. 161, 2015.

[12] B. Morschheuser, J. Hamari, J. Koivisto *et al.*, "Gamified crowdsourcing: Conceptualization, literature review, and future agenda," *International Journal of Human-Computer Studies*, vol. 106, pp. 26–43, 2017.

[13] G. F. Tondello, H. Premsukh, and L. E. Nacke, "A theory of gamification principles through goal-setting theory," in *HICSS*. IEEE, 2018.

[14] J. Hamari, J. Koivisto, and H. Sarsa, "Does Gamification Work? - A Literature Review of Empirical Studies on Gamification," in *HICSS*. IEEE, 2014, pp. 3025–3034.

[15] W. M. P. van der Aalst, A. Adriansyah, A. K. A. de Medeiros *et al.*, "Process Mining Manifesto," in *BPM Workshops*, ser. LNBIP, vol. 99. Springer, 2011, pp. 169–194.

[16] R. J. C. Bose, R. S. Mans, and W. M. P. van der Aalst, "Wanna Improve Process Mining Results - It's High Time We Consider Data Quality Issues Seriously," in *CIDM Symposium*. IEEE, 2013, pp. 127–134.

[17] A. Gal, A. Senderovich, and M. Weidlich, "Challenge Paper: Data Quality Issues in Queue Mining," *Journal of Data and Information Quality*, vol. 9, no. 4, pp. 18:1–18:5, 2018.

[18] M. Jans and P. Soffer, "From relational database to event log: Decisions with quality impact," in *BPM Workshops*. Springer, 2018, pp. 588–599.

[19] X. Lu and D. Fahland, "A Conceptual Framework for Understanding Event Data Quality for Behavior Analysis," in *ZEUS*, ser. CEUR, vol. 1826, 2017, pp. 11–14.

[20] A. Koschmider, M. Ullrich, A. Heine *et al.*, "Revising the Vocabulary of Business Process Element Labels," in *CAiSE*, ser. LNCS, vol. 9097. Springer, 2015, pp. 69–83.

[21] N. Tax, E. Alasgarov, N. Sidorova *et al.*, "Generating Time-Based Label Refinements to Discover More Precise Process Models," *J. of Ambient Intelligence and Smart Environments*, vol. 11, no. 2, pp. 165–182, 2019.

[22] S. J. van Zelst, M. F. Sani, A. Ostovar *et al.*, "Detection and removal of infrequent behavior from event streams of business processes," *Information Systems*, vol. 90, p. 101451, 2020.

[23] H. van der Aa, A. Gal, H. Leopold *et al.*, "Instance-Based Process Matching Using Event-Log Information," in *CAiSE*, ser. LNCS, vol. 10253. Springer, 2017, pp. 283–297.

[24] F. Pittke, H. Leopold, and J. Mendling, "Automatic detection and resolution of lexical ambiguity in process models," *IEEE Transactions on Software Engineering*, vol. 41, no. 6, pp. 526–544, 2015.

[25] M. Becker and R. Laue, "A comparative survey of business process similarity measures," *Computers in Industry*, vol. 63, no. 2, pp. 148–167, 2012.

[26] M. Weidlich, T. Sagi, H. Leopold *et al.*, "Predicting the quality of process model matching," in *BPM Conference*, ser. LNCS, vol. 8094. Springer, 2013, pp. 203–210.

[27] C. Klinkmüller, H. Leopold, I. Weber *et al.*, "Listen to me: Improving process model matching through user feedback," in *BPM Conference*. Springer, 2014, pp. 84–100.

[28] E. Scibona, "Cost-effective and Scalable Activity Matching using Crowdsourcing," Master's thesis, Politecnico di Milano, Italy, 2018.

[29] Y.-K. Chou, *Actionable gamification: Beyond points, badges, and leaderboards*. Packt Publishing Ltd, 2019.

[30] S. Sadeghianasl, A. H. M. ter Hofstede, M. T. Wynn, and S. Suriadi, "A contextual approach to detecting synonymous and polluted activity labels in process event logs," in *CoopIS*, ser. LNCS, vol. 11877. Springer, 2019, pp. 76–94.

[31] G. Zichermann and C. Cunningham, *Gamification by design: Implementing game mechanics in web and mobile apps*. O'Reilly Media, 2011.

[32] L. Von Ahn, B. Maurer, C. McMillen *et al.*, "reCAPTCHA: Human-based Character Recognition via Web Security Measures," *Science*, vol. 321, no. 5895, pp. 1465–1468, 2008.

[33] M. S. Hardas and L. Purvis, "Bayesian vote weighting in crowdsourcing systems," in *ICCCI*, ser. LNCS, vol. 7653. Springer, 2012, pp. 194–203.

[34] van Dongen B.F., "BPI Challenge 2019. 4TU.ResearchData. Dataset." https://doi.org/10.4121/uuid:d06aff4b-79f0-45e6-8ec8-e19730c248f1.

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

[36] P. Sweetser and P. Wyeth, "Gameflow: a model for evaluating player enjoyment in games," *Computers in Entertainment*, vol. 3, no. 3, p. 3, 2005.

[37] L. J. Cronbach, "Coefficient alpha and the internal structure of tests," *Psychometrika*, vol. 16, no. 3, pp. 297–334, 1951.

An Expert Lens on Data Quality in Process Mining

Robert Andrews*, Fahame Emamjome*, Arthur H.M. ter Hofstede* and Hajo A. Reijers†
*Queensland University of Technology, Brisbane, Australia
{r.andrews,f.emamjome,a.terhofstede}@qut.edu.au
†Utrecht University, Utrecht, The Netherlands
h.a.reijers@uu.nl

***Abstract*—The success of a process mining project is highly dependent on the quality of the event log data, the degree to which quality issues are detected, and the way they are resolved. The detection and resolution of data quality issues requires a systematic approach that is aware of the organisational context in which event log data is created. To this end, the Odigos framework has been developed in prior work. The focus of this paper is the validation of this framework through semi-structured interviews with a range of experts in process mining. The experts confirmed the utility of the framework, provided valuable insights into data quality in practical settings, and suggested enhancements to the Odigos framework.**

***Index Terms*—process mining, data quality, Odigos framework, expert validation**

I. Introduction

Event log quality is a critical success factor in any process mining project [1]. Process mining project methodologies that refer to event log quality [2]–[4] make provision for quality issues to be addressed primarily in the pre-processing phase. This has the caveat that further consideration may be required in later phases as event logs are manipulated to address different types of analysis. Curiously, such methodologies pay scant attention to practical elements such as the identification of data quality issues, the role of data quality in guiding event data extraction and log construction, and the impact of low data quality on process mining analyses [5]. In many process mining case studies, researchers limit data pre-processing to merely transforming raw event data to a format that can be consumed by process mining tools, and to uncritically report analysis outcomes. We refer to this *garbage in - gospel out* effect as *naive* process mining [6]. As is pointed out in [5], identifying the root causes of quality issues in event logs helps researchers to deal effectively with quality issues and, thence, to derive informed insights from their analysis. However, existing approaches to data quality and log cleaning (e.g. [7], [8]) are more focused on treating data quality symptoms (in a given log) than on recognising the root causes of those issues. In [6], the authors propose the notion of *informed* process mining, which involves a consideration of the context in which a process executes as a means of identifying root causes of event log quality issues. To deal with data quality issues in a systematic way, the *Odigos* framework [9] guides process mining researchers in diagnostically and prognostically identifying data quality issues in process-related event data. The main contribution of this paper is a qualitative, interview-based, expert evaluation of the Odigos framework, which confirmed the utility of the framework. Additionally, interaction with the experts (i) enhanced our understanding of the problems with data quality issues and awareness around data quality issues, and (ii) resulted in suggestions for refining the framework in the light of their collective experiences. The remainder of this paper is organised as follows. In Section II we discuss unresolved issues relating to event-data quality, which serve to position the Odigos framework. To make this paper self-contained, we briefly describe the main features of the Odigos framework in Section III. In Section IV we describe our approach to evaluating the relevance and usefulness of the Odigos framework through expert interviews.The main results are presented in Section V. In Section VI we discuss some important themes that emerged through analysis of the interviews. In Section VII we discuss possible limitations of our sampling approach as well as insights arising from the evaluation and directions for future work.

II. Related work

The importance of data quality for data analysis in general, and for process mining in particular, is evident [10]–[12]. Process mining methodologies such as PDM [2], L* [3], and PM^2 [4] do not provide methodological guidance for the identification of quality issues in process mining studies, and apart from [10], there is little awareness of the impact of data quality on the findings of process mining studies [5]. Event log data quality frameworks, such as those described in [10] and [1] are useful for labeling identified symptoms of data quality. However, neither helps with identifying the causes of those quality issues, nor with recommending possible remedies. Further, the approach in [1] is rooted in hospital information systems, is not generalised, and as a result does not inform researchers in approaching data quality in domains other than healthcare. In [11], the authors, based on their experience in dealing with multiple event logs, abstract a set of commonly occurring event data quality issues as pattern templates which link the manifestation of each data quality issue to likely underlying causes. The authors show that data quality issues can be detected by searching for the relevant pattern's signature in the event log. They also discuss the impact on a process mining analysis of each pattern and suggest possible remedies for the detected quality issues. The paper provides valuable insights regarding some common root causes of data quality issues. While they may not span the entirety of event log data quality issues, it is a step towards a systematic, generalisable

978-1-7281-9832-3/20/$31.00 ©2020 IEEE
DOI 10.1109/ICPM49681.2020.00018

approach to dealing with data quality issues. Preliminary work towards operationalising this approach was given in [13]. The metrics-based approach to assessing data quality during the preparation and planning stage of process mining projects in [14] highlights how data quality issues can affect the findings of a process mining analysis. The authors argue that identifying the root causes of quality issues prior to conducting process mining analysis and engagement with stakeholders can provide insights to possible remedies for the quality issues but do not support this with methodological guidance. The Care Pathways Data Quality Framework (CP-DQF) [15] uses the quality framework described in [10] to support systematic management (identification, recording, mitigation, reporting) of data quality issues affecting process mining research using electronic healthcare records (EHR). CP-DQF provides a comprehensive list of data quality issues in the context of EHR; however, this framework pays limited attention to uncovering root causes of quality issues and is constrained by its exclusive focus on EHR systems. The 3x3 DQA framework described in [16] also deals with requirements of EHR data from a reuse perspective. The framework takes the relationship between task dependence (fitness for use), data dimensionality and data quality assessment as its core feature. The 3x3 DQA framework is, as the name suggests, a matrix with Task Dependence (broken into three data constructs - completeness, correctness and currency) on one axis, and Data Dimensionality (broken into Time, Variables and Patients) on the other axis. Each cell is operationalised by one or more metrics with the determination of what constitutes "sufficient quality" made by the user in the light of the user's "understanding of the data, the clinical phenomena being examined, and the methods of analysis used" [16, p. 9]. This framework was assessed by expert interview with results of quantitative and qualitative assessments of responses suggesting that the framework was not intuitive, that the operationalised constructs of EHR data quality need to be improved and the cognitive overhead of interpreting the complex guideline logic was a barrier to practical use. The fact that the framework is specific to EHR data could be considered a further limitation.

III. Background - Odigos Framework

The Odigos framework (see Figure 1) suggests a systematic approach to dealing with data quality issues in event logs in both diagnostic and prognostic ways. Based on the principles of Semiotics [17], the Odigos framework provides a systematic understanding of the context of process mining. Semiotics is the study of signs, their creation and how they generate meaning. Almost everything that we interact with and is capable of generating some meaning can be a sign. Figure 1 shows the semiotic content, i.e. actual processes, event data and event logs, in the center of framework. The Odigos framework [9] defines the Social world in terms of both organisational context (situational) and wider social context beyond the organisation (macro) that can influence people and IT systems. The Material world in the context of process mining refers to all IT systems used to support the process. The

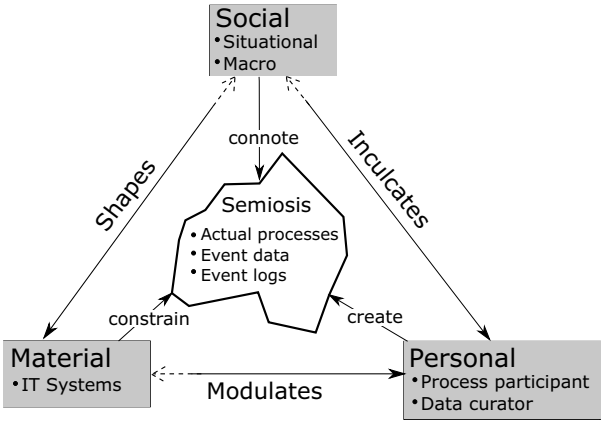


Fig. 1. The Odigos data quality framework.

Odigos framework identifies the different layers (presentation, application, and data) of IT systems as being important when considering data quality issues. The Personal world includes the Process participants who carry out processes, and the Data curator who is responsible for generating event logs from the recorded event data for use by analysts. The central tenet of the Odigos framework is that the semiotic content (processes, event data, event logs) results from direct actions arising from the individual worlds (indicated by the connote, create, and constrain arrows in Figure 1) which are ultimately driven by interactions between the worlds (indicated by the Inculcates, Modulates, and Shapes arrows in Figure 1). Thus, the Odigos framework captures the process mining context. By understanding the context, it is possible to understand/explain the semiotic content, including data quality. In this paper, we refer to the interactions between worlds and semiotics using 'paths' (see Table I for a complete listing). For example, path 4, the interaction between the Material (IT system) world Modulating the Personal world (Process participants) influencing semiotics (recording event data), could explain why values in a particular field do not bear any relation to the label or database column definition. Process Participants have employed a 'workaround' to capture process-critical data with an IT system design that does not properly support the process.

IV. Methodology

The Odigos framework [9] provides a high level conceptualisation of the process mining context. It aims to help process mining researchers discover the root causes of quality issues in event logs. However, for such a framework to be successfully applied in process mining methodologies, further steps are required to confirm the usability and relevance of the framework. In line with principles given in [18] and [19] a qualitative approach, using semi-structured interviews with process mining experts, was used to explore the usability and relevance of the Odigos framework, and purposive (or judgement) sampling was used to select participants. Interviews

are one of the main data collection methods in qualitative research and purposive sampling is relevant as insights and expert knowledge to properly consider the framework is held by only certain members of the process mining community. The sample included researchers and practitioners deemed to be experts in the field of process mining and data quality.

A. Expert Profile

We selected a panel of 15 experts of whom 11, coded as E1..E11, agreed to participate. Of the 4 who did not participate, 2 said they were unable to participate due to time constraints and 2 did not reply to the invitation. Selection criteria included: (i) demonstrated experience in practical applications of process mining, e.g. published case studies or employment in a senior role related to process mining; and (ii) demonstrated interest in event log data quality, e.g. publications in this area; or (iii) recommendation from one or more of the selected experts. The experts (a) were distributed across the globe – 3 continents and 6 countries, (b) have a combined research and practice experience of approx. 130 years (average experience = 10.25 years), (c) work across multiple domains (the predominant application area being healthcare followed by education and manufacturing, and (d) have, collectively, 240 process mining publications and 12 (refereed) publications relating to event data quality[1].

B. Interview Process

We used a semi-structured approach in designing the interview protocol [20]. As our aim was to gauge the usefulness and relevance of the Odigos framework through linking to the experts' experience, our interview approach was based on a protocol that was pretested to ensure questions would be understood and properly interpreted, would yield the appropriate objectives, and would be scoped to encourage open-ended input. The interviewer ensured that all questions in the protocol were covered during the interview. However, digressions to related topics of discussion were permitted in order to increase the richness of the information captured. The interview included an introduction to the Odigos framework through sets of examples. Participants were free to ask questions or comment during this time. Next, the interview continued using three different sets of questions. The first set of questions focused on the specific role of the Social, Personal and Material worlds (inner arcs of the framework) in creating data quality problems. The second set of questions examined the interactions between worlds (outer arcs of the framework) and their role in creating quality issues in an event log. The third set of questions were about the overall relevance and potential implications of the framework for a process mining project.

C. Interview Analysis

We combined both quantitative and qualitative approaches in analysing the interviews. The interview questions required the experts to rank the causal paths of influence in terms of usefulness, and also to provide examples from their experience. Using the usefulness ranking in combination with the number of examples in each given arc (Table I), we derived, in the first stage of our analysis, an initial quantitative estimation of the usefulness of each arc. Qualitative analysis of the responses provided a richer understanding of the experts' views on each causal path and the importance of the arcs in explaining data quality issues. The qualitative analysis was supported by Nvivo as a data analysis and evidence management tool and involved coding the interviews inductively to identify emerging themes. The coding scheme was based on the Odigos framework and the structure of the questionnaire (discussed in Section IV-B). Using open and axial coding approaches, we then identified 76 unaggregated codes representing emerging themes within and beyond the Odigos framework's original elements. The themes that emerged in relation to the causal paths are discussed in more detail in Section V and relate to the Social, Personal, and Material worlds of the framework. The themes discussed in relation to the outer arcs are represented in Table I. The themes which were not within the scope of the Odigos framework are further discussed in Section VI, Emergent Themes.

[1]Based on data from https://dblp.uni-trier.de/

V. The validation results

The first set of interview questions revolved around the issue as to whether the three worlds of the Odigos framework cover the possible direct causes of data quality problems as observed by the experts. Almost all the experts found the list of presented direct causes to be comprehensive. Further, they were able to elaborate on the Social, Personal, and Material worlds in contributing to data quality issues as experienced by them.

A. Social World

The importance of the Social world, and more specifically, the organisational context, was highlighted by all the interview participants. The participants, based on their individual experiences, mentioned the following elements of the social world resulting in data quality problems in event logs:

- Organisation culture (E2, E4, E5). The same system with the same functionalities is used differently in different organisations.
- Social pressures and agreements in the work environment (for example in a hospital emergency department, where the highest priority is delivering treatment, with less attention being paid to recording tasks in the IT systems) (E1, E8)
- Management style (having a process manager role will decrease data quality issues) (E5)
- Organisation structure (focus on specialisation rather than workflow and the processes) (E5). E5 also mentioned that in an organisation with matrix structure management in comparison to a hierarchical structure management, there is a lower chance of having data quality problems in event data. "[...] usually in a matrix structure you have "vertical" department managers and "horizontal" process

Path of Influence					Usefulness			#Exampl.	Themes
Path	Influencing model element	Model arc	Model element influenced	Semiotic content	VU	SU	NU		
1	SOC	Inculcates	PP	Process execution	8	3	0	7	Cultural differences, freedom in doing tasks, regulations and performance criteria, temporal social processes
2	IT	Modulates	PP	Process execution	8	2	1	2	IT Systems designed to capture information rather than supporting the workflow
3	SOC	Inculcates	PP	Recording event data	8	3	0	9	Performance criteria, registration after fact, terminologies and language, task priorities, organizational norms and training
4	IT	Modulates	PP	Recording event data	10	1	0	8	IT system not supporting the record of processes, human error as a result of system design
5	SOC	Inculcates	DC	Extracting event logs	9	2	0	3	Data curators lack of knowledge about process mining, lack of knowledge of domain, privacy concerns
6	IT	Modulates	DC	Extracting event logs	10	0	1	1	Interoperability between systems and extracting tools, data curator knowledge and skills in extracting, complexity of databases and vendor power over data extraction, integrating the logs
7	SOC	Shapes	IT	Recording event data	8	3	0	9	Different terminology and codes for identical system artefacts, performance criteria in design of systems, IT systems designed with goals other than workflow management, organisational silo structure
8	PP	Modulates	IT	Recording event data	7	3	0	5	Manual entries, deviation of the standard process path, misusing admin access
9	DC	Modulates	IT	Recording event data	6	4	1	0	Data curator role vs data admin

TABLE I
USEFULNESS: VU=VERY USEFUL; SU=SOMEWHAT USEFUL; NU=NOT USEFUL AT ALL
INFLUENCING MODEL ELEMENT: SOC=SOCIAL ORGANISATIONAL CONTEXT; IT=IT SYSTEMS; PP=PROCESS PARTICIPANTS; DC=DATA CURATOR
MODEL ELEMENT INFLUENCED: IT=IT SYSTEMS; PP=PROCESS PARTICIPANTS; DC=DATA CURATOR

managers. By granting mandate and budget explicitly also to a process owner it could be more likely that processes, workflow and event logs get more and better attention. Thus raising the chance of better data quality." (E5)[2]

- Performance criteria and reward systems with a process-focus. "The company did not have target on process execution. They were not organised to reward people based on processes" and "The reward system also can influence on the execution and record of the processes and creation of event logs". (E5)
- Legal requirements: including performance regimens or government privacy regulations. (E2, E4, E7)
- Cultural differences between countries (E4). For example, the same processes, using the same systems, would be done differently in different countries with different working cultures.
- Consideration of time and temporal nature of business processes in the framework. Processes may change faster than the IT systems which support these processes. "Processes are in a state of flux" and "in process mining processes are considered static". (E6)

B. Personal World

The Odigos framework recognises two roles as part of the Personal world: Process participants and Data curators. While the experts acknowledged these roles, they mentioned a richer set of relevant roles, including:

- Process designers. They configure the processes in the systems. "If they use the wrong label for the processes and put wrong data, or use inconsistent naming or they forget to log some certain data [...] or if they enter a process with a typo and correct it later". (E2)
- Process analyst. "The way [the] process analyst asks for an event log has [an] influence on what they get and the meaning they generate from it. And how the process analyst looks at data". (E5)
- Customers. For example, patients or loan applicants if they directly enter data. (E3)
- Data administrator or IT team. It was argued that this role should be separated from the role of data curator. (E2, E4, E6)
- Data curators are not necessarily lone operators, but can form a "chain". (E6, E7)
- Bots and intelligent systems. "So it is worth to think of IT systems as a new actor, they are not only collecting data. They are also an actor involved, even more important than the participants". (E6)

Almost all of the experts agreed with the importance of these roles in terms of root causes of data quality issues. One of our eleven experts (E9), however, suggested that the role of process participants in creating data quality issues can be minimised by better designing IT systems to avoid, for example, human errors and control [flow] variation of the processes.

C. Material World

All of the experts agreed on the important role IT systems play in causing data quality issues in event logs. They men-

[2]Quoted from the expert's email, in response to our request for feedback on the paper, 21st of August 2020.

tioned, based on their own experiences, the following data quality considerations in relation to IT systems:

- Interoperability between different systems in use, and the problem of linking data from multiple sources. (E1, E6, E7, E8)
- The interoperability between systems in use and the tools used by data curators to derive the data (data curation systems). (E1)
- Information architecture and the design of applications (e.g. system or application errors) (E3, E6). One of the experts (E6) also mentioned that data quality issues could be created as a result of an error in linking applications in the systems, and gave the example of a hospital system that creates automatic (ward) bed requests at 4am when the emergency department interface is connected. If the ED system has a bed request without the request time field entered, the bed request system automatically populates the field with the current time (4am).
- Mixed granularity of events and timestamps recorded. (E1, E11)
- The compatibility of IT systems in recording event data with the requirements of process analysis (E5, E11) (for example the appropriate granularity of time stamps). "The IT systems are only focused on a piece of information but not the entire system and time in which value is created" and "Most systems are not designed to capture event logs". (E5)
- Timing of system updates by software vendor. (E6)

D. Different Paths of Influence

In the second section of the interview, questions focused on the nine different causal paths in the Odigos framework relevant to explaining the root causes of data quality issues in event logs. Our purpose firstly was to identify whether participants perceive these causal paths as useful in explaining data quality issues (useful referring to if they had seen data quality issues which could be explained using these causal paths) and, secondly, whether they could provide any examples that would help us to further specify these causal paths in our framework. Table I summarises the results of the interviews on these nine causal paths. Each row in the table refers to a single causal path and (i) summarises the experts' perceptions of the usefulness of the causal path in explaining the root causes of data quality issues, (ii) gives the number of examples of data quality issues that they experienced in terms of this path, and (iii) provides a thematic analysis of the discussion surrounding experts' experiences related to the path. Table I shows that experts saw the nine causal paths as (predominantly) *very useful* or *somewhat useful* in explaining data quality issues. In the following sections we discuss the main themes that emerged and the examples provided in relation to each path.

Path 1: Social-process participant in process execution Table I shows that eight out of eleven of our experts agreed that path 1 is a very useful causal path in explaining data quality issues. The seven examples given by the interviewed experts highlighted the importance of organizational culture (different ways of doing the same process with the same system (E2), freedom in doing the tasks and not adhering to any formal processes (E5), deviation from formal processes because of customers' pressure (E11)), temporal social processes (for example change of the processes during night vs day shifts (E6) or use of new drugs by physicians when those drugs are not yet defined in the systems (E3)) and performance criteria (in an emergency department, patients who could be seen earlier were made to wait close to 4 hours (E6)).

Two experts (E3, E10) raised concerns about differentiating between data quality issues and infrequent aberrant process behaviour. We will further explore this in Section VI.

Path 2: IT systems-process participants in process execution Table I shows that eight of the eleven interviewed experts agreed that this path is very useful, three ranked it as somewhat useful, and one participant commented that this is not useful according to their experience with data quality issues. Three of the interviewees mentioned that IT systems (both applications and interfaces) not supporting the business processes are the main cause for data quality issues in this category (E5, E9, E10). For example in a loan process, the system did not allow multiple offers for the same client and every time a client accepted an offer the other offers needed to be deleted from the system manually, only then the loan offer could be confirmed (E4).

Path 3: Social-process participants in recording event data Eight out of eleven experts agreed that path 3 is very useful in explaining data quality issues and nine different examples were given by the experts. Two of the common themes between different respondents were registration after the fact and batch recording (E2, E4, E5, E8, E9, E10). Some experts (E1, E4) provided several examples that reflected that process participants had to deal with different social pressures and priorities (e.g. in healthcare, patients and their treatment are prioritised, in a mortgage management context focus is on customer service) rather than with recording tasks in systems. Other examples provided by the experts reflected the role of performance criteria in the way process participants record the data (E1, E5). For example in a manufacturing context, maintenance personnel's performance is measured according to the number of repairs they complete or attend to. Consequently, these process participants did not pay attention to recording the activities accurately during each case, but only logged these for a case after they finished it (E1). Experts also mentioned organisational norms and training in the use of the IT systems as another factor affecting process participants' recording of the tasks (E1, E8). One of the main related examples mentioned was medical staff in hospitals and their resistance to using IT systems (E1, E6, E8, E9).

Our experts emphasised the importance of designing systems that support recording the workflow and guide the process participants through processes. They also emphasised having performance criteria which are not inconsistent with workflow management (E5).

Path 4: IT systems-process participants in recording event data Almost all experts (ten out of eleven) agreed that path 4

is very useful in explaining quality issues in event logs. In relation to this path the experts mentioned that IT systems are not supporting the processes (E1, E4, E5, E8, E11). Different reasons were identified for this lack of support including (i) database design (E4), (ii) terminologies used in systems being different from the terminologies used by process participants (E8), (iii) lack of automation allowing human errors in entering data (E1, E2), as one of the experts mentioned "sometimes they [process participants] have to log as an extra activity, sometimes they log when they save, if the system is focused on workflow then the logs are recorded automatically" (E5), and (iv) not having a user friendly interface. For example, in a hospital using SAP, users had to manually enter data. However, there were many separate forms and data fields, many of which were mandatory. This caused users to enter one word or a few letters in each field to allow progression through the form, perhaps assuming that they will complete it later (E8, E9).

Path 5: Social-data curators in extracting event logs This path was considered very useful by nine out of eleven experts, two of them ranked it as somewhat useful, with three examples being given by them. Themes relating to data curators included (i) lack of knowledge of process mining requirements (E1, E2, E4, E8, E10), (ii) having domain knowledge (E2, E7), and (iii) privacy concerns (E3, E4, E5, E6, E7, E8). Examples of data quality problems were anonymisation of data in such a way that it conflicts with the aims of process mining (E5), for example privacy concerns leading to the changing of timestamps (possible re-identification of patients) (E6) or the aggregation of data without separate case IDs (E4).

Path 6: IT systems-data curators in extracting event logs This path was considered very useful by ten out of eleven experts. They mentioned different factors including data integration problems between different IT systems (E1, E3, E6, E7), different clocks or time settings of different devices (E11), interoperability issues between IT systems and data extraction tools (for example extracting data to the old version of MS-Excel which ended up in many missing rows (E1)), complexity of data extraction and software vendors' power over this (E8), chain of data curators with consequent data provenance chain (E6, E7). "It can be difficult for data curators to link data from different source systems so they may end up doing probabilistic rather than deterministic linkage" (E6).

Path 7: Social-IT systems in recording event data The influence of the social context on IT systems was considered as very useful by eight out of eleven process mining experts with nine examples given in relation to this path. Participants mentioned several factors including the impact of performance criteria in designing the systems (E2, E5), IT systems designed with goals other than workflow management (E4, E9), siloed organisations where data is compartmentalised and multiple disconnected systems are used by each department (E7, E8), various terms used within the organisation to refer to the same concept (E1), and IT systems designed with i) varying timestamp configurations (future timestamp is assigned as a default value (E11)) and ii) different case ID format rules (in a call centre a dummy case ID is assigned by a call centre agent and has the same format as a customer number (E11)). Examples include using performance measures for patients' length of stay in emergency departments in hospitals (E6) or defining a KPI around resolution time for IT tickets (E2). In both scenarios, process participants exploited the system configuration to meet the defined KPI without actually achieving those KPIs.

Path 8: Process participants-IT systems in recording event data Process participants influencing the recording of event data in IT systems was considered as useful by seven out of eleven experts, with five examples provided. In relation to this path, the experts mentioned factors such as audit functionality which can be activated by process participants (E7), misusing admin access to the system (E2, E10) which results in deviation from the standard process path, and manual data entries by process participants (E4).

Path 9: Data curators-IT systems in recording event data The influence of data curators on IT systems design was considered useful by six out of eleven experts. A number of them felt that in the definition of path 9, the role of data curator should be changed to data administrator (E2, E4, E6). One of the experts mentioned that data quality issues resulting from path 9 could also be explained by path 7 (E4). One expert stated that it is important for a process analyst to acquire knowledge (either from data curator or data admin) about recent/planned changes and updates to IT systems prior to asking data from data curators (E8).

E. Overall relevance and effectiveness

We asked the experts to rate the overall framework in terms of 1) its effectiveness in providing explanations for data quality problems in process mining, and 2) the relevance of the approach in improving process mining methodologies in the data pre-processing stage. All participants agreed that the framework is *very effective* or *somewhat effective* in explaining quality issues and that it is *very relevant* or *somewhat relevant* for the pre-processing stage. Expert E11 suggested that the framework is somewhat effective and it needs to be explained with examples for someone "to get into thinking [about] the model". In relation to the relevance of the framework in the data pre-processing stage of process mining methodologies, one of the experts (E4) said "It is a must [...], there is no other approach in existing process mining studies". Experts also mentioned that the framework (i) helps to ask the right questions (E2, E8), (ii) provides the landscape of data quality issues (E6), (iii) helps researchers to look into the right place (E6, E5), and (iv) assists with finding the actual problem (E11). "Also you can identify the questions that you can ask from data curator and anticipate problems, gives complete sets of questions" (E8). Four of the process mining experts (E1, E3, E6, E11) mentioned that the Odigos framework can be used to classify data quality issues.

Experts also suggested that the framework would be more useful if it were more detailed (E1, E2, E7). Three of them suggested that the framework should prioritise data quality

issues, i.e., what are the quality issues that affect the process mining findings (E2, E3, E10). All experts agreed with the statement that there is need for a supporting methodology and one expert said that the Odigos framework can be used to improve existing data extraction and pre-processing methodologies (E8). Overall, we found strong support from all of the process mining experts on the effectiveness and relevance of the Odigos framework.

VI. Emergent Themes

Here we discuss themes that were not anticipated by the interviewers but emerged from the interviews. We reflect on how they will be considered in the further development of the Odigos framework.

New roles As we discussed in relation to the personal world (section V-B), experts introduced new roles involved in the creation of event data and the introduction of data quality issues. These new roles include the process designer (E2), process analyst (E5), customer as process participant (E3), data administrator (E2, E4, E6), chain of data curators (E6, E7), (i.e. where there is more than one data curator involved in extracting event logs), and intelligent systems such as bots (E6). The Odigos framework definition of 'process participant' is not exclusive to employees, so any role which is involved in recording data and completing tasks using the IT systems, such as a customer or patient, can be considered in this definition. We agree with expert E2 that the process designer role should be considered in relation to data quality issues. Rather than only this addition, we argue that it is then also important to include the role of formal processes and how processes are defined and embedded in IT systems, in the definition of the Material world. As one of our experts (E5) mentioned, the role of process analyst is important at the point that they are asking for data and when they are seeking meaning from the data within the domain context. Accordingly, the process analyst is a user of the Odigos framework. Therefore, we do not need to explicitly consider this role in the definition of the Personal world. We agree with experts E6 and E7 that the data curator could be more than one person. Accordingly, we now extend the definition of data curator to include any person or groups of people who are involved in extraction of data and transferring it to the form of event log(s) for the process analyst. One of our experts (E6) mentioned the emerging role of bots and intelligent systems in the creation of event data and data quality issues in event logs. Intelligent systems and bots could be considered in the Odigos framework in terms of the material world. Advances in technology (material world) may change, but will not diminish, the role and significance of process participants and social structures.

What is data quality? Several experts raised the question of what actually constitutes a data quality issue, and, in particular, if data representing (possibly infrequent) variations in process behavior should be defined as a data quality problem (E2, E3, E6 and E10). "I have [the] more philosophical question whether you need to include the processes. If people are doing the processes in a different way, is that a quality issue?" (E10). The original motivation behind the Odigos framework was to look beyond symptoms of data quality issues (meaning any anomaly in a data set which hinders the standard analysis processes using process mining tools). We argue that any data quality symptom in an event log is potentially representing a reality. It should be investigated to decide how it can be dealt with. We agree with E6 that "we should get rid of the mindset that data quality is an inconvenience that should be done quickly to get to the analysis phase, exploring data quality is about understanding data".

Technology advancement Technological advancement could either improve data quality issues or introduce new ones. Expert E7 mentioned that causal paths 1–4 would be less important with technology development and automation. "At the moment [it] is a problem but this is a problem where technology is not developed. For example, in banks you would not have that problem but in healthcare it is always behind in terms of technology" (E7).

Solutions to data quality problems Experts mentioned several solutions to existing data quality problems. Having process-aware information systems and designing systems which guide process participants was one of the main solutions (E2, E4, E5, E9). "The big task is to ask designers and business managers to have the right metrics and to design process-aware systems." (E5). Expert E9 was of the opinion that in less stressful work environments, a process-aware system with a well designed user interface would improve recording of the tasks with consequent improvement in the quality of event data. However, in contexts with higher work pressure, expert E9 felt that merely improving the IT systems would not solve the data quality problems. More clarity about privacy requirements imposed by government regulations and using better encryption methods was another factor mentioned by expert E7. The importance of communication between process analyst and data curator was added by expert E9 as a solution to deal with data quality issues: "It is important to say what we [process analyst] need otherwise they [data curators] just give us a chunk of data. Maybe some issues can be fixed by them easily if we clarify what do we need" (E9).

VII. Discussion

In this paper, we validated the Odigos framework, an approach for identifying causes of data quality issues affecting process mining. We interviewed eleven process mining experts and, through these interviews, gained additional insights into data quality and the usefulness of the Odigos framework. For this study, purposive sampling was chosen with a view of reaching a targeted sample quickly and because, in this instance, sampling for proportionality was not the primary concern. We note that such an approach increases the likelihood of getting the opinions of the target population at the possible expense of overweighting opinions of subgroups in the target population that are more readily accessible. We also note that as the interviews with experts were conducted over a period of several weeks, the conduct of the interviews changed

slightly as we got more familiar with the process and better at explaining concepts underlying the framework to the experts.

Overall, the framework was perceived as useful by the experts who confirmed that such a framework, together with a supporting methodology to guide process mining researchers to go beyond data quality symptoms, fills a gap in the field. Our discussions with the experts unearthed interesting experiential examples, which provided further insight into the causal paths of the framework. They also broadened our perspectives on the three worlds of the framework.

An interesting insight that followed from an interview with one of the experts was that the removal or mitigation of the root cause of a data quality issue is contingent upon the organisational context and thus on an understanding of the causal paths and their ends (the worlds involved).

In some of the examples provided by the experts, we could see that both the Social and Material worlds played a role in the way process participants use the IT systems. To identify what is the best mitigation plan in these scenarios, careful consideration should be paid to the specific context and how plausible changes to these worlds would impact on the way IT systems are used. For example, in a hospital emergency department, the work pressure and the social agreements on prioritising patient welfare is non-negotiable. This may mean that mitigation measures are more likely to be successful (in this organisational context) if changes can be made in the material world, i.e. having IT systems that can accurately and automatically capture activities without compromising process participants' social world obligations. The feasibility of this solution, however, needs to be assessed against other social world considerations. One can think of implementation cost, risk management, training provision in the use of current IT systems, process redesign, etc. This may then determine the 'balance point' of mitigation efforts. That is, emphasis towards one end of the causal path with some effort applied at the other. Thus, the Odigos framework facilitates development of mitigation measures that address root causes of data quality issues tailored to a given organisational context. In the same vein, the recognition that the context at either end of a causal path has changed may be a trigger to (prognostically) re-examine the impact of the change on data quality with early revision of mitigation efforts.

In some of the examples given by the experts, we found multiple causal paths from the Odigos framework that explained the problems. However, we could argue that in any context one of these has the greatest explanatory power. We also note that, it is possible that causal paths may be 'chained', which may reflect an ancestor sequence of root causes.

Given the acceptance of the Odigos framework by the experts, future work will focus on development of methodological guidance in the use of the framework for purposes including (i) *diagnostic use* - identifying root causes of identified data quality issues, (ii) *prognostic use* - assessing the organisational context to anticipate data quality issues, (iii) *remediation* - devising data cleaning strategies to rectify data quality issues in such a way that the goals of the process mining analysis are not compromised, and (iv) *prevention* - using the framework to understand the organisational context and develop prevention strategies that best fit with the imperatives of, and interactions between each of the social, material, and personal worlds.

ACKNOWLEDGEMENT

The authors would like to thank the process mining experts for their valuable feedback and for making time for the interview in their busy schedules.

REFERENCES

[1] R. Mans, W. van der Aalst, R. Vanwersch, and A. Moleman, "Process mining in healthcare: Data challenges when answering frequently posed questions," in *Process Support and Knowledge Representation in Health Care*. Springer, 2013, pp. 140–153.

[2] M. Bozkaya, J. Gabriels, and J. van der Werf, "Process diagnostics: a method based on process mining," in *Int. Conf. on Information, Process, and Knowledge Management*. IEEE, 2009, pp. 22–27.

[3] W. van der Aalst, A. Adriansyah, A. De Medeiros, F. Arcieri, T. Baier, T. Blickle *et al.*, "Process mining manifesto," in *Int. Conf. on BPM*. Springer, 2011, pp. 169–194.

[4] M. van Eck, X. Lu, S. Leemans, and W. van der Aalst, "PM^2: A process mining project methodology," in *CAISE*. Springer, 2015, pp. 297–313.

[5] R. Andrews, M. Wynn, K. Vallmuur, A. ter Hofstede, E. Bosley, M. Elcock, and S. Rashford, "Leveraging data quality to better prepare for process mining: An approach illustrated through analysing road trauma pre-hospital retrieval and transport processes in Queensland," *IJERP*, vol. 16, no. 7, p. 1138, 2019.

[6] F. Emamjome, R. Andrews, and A. ter Hofstede, "A case study lens on process mining in practice," in *OTM Confederated Int. Confs "On the Move to Meaningful Internet Systems"*. Springer, 2019, pp. 127–145.

[7] R. Bose and W. van der Aalst, "Trace alignment in process mining: Opportunities for process diagnostics," in *Int. Conf. on BPM*. Springer, 2010, pp. 227–242.

[8] H.-J. Cheng and A. Kumar, "Process mining on noisy logs–can log sanitization help to improve performance?" *Decis. Support Syst.*, vol. 79, pp. 138–149, 2015.

[9] F. Emamjome, R. Andrews, A. ter Hofstede, and H. Reijers, "Alohomora: Unlocking data quality causes through event log context," in *Proce. 28th Eur. Conf. on Inf. Syst. (ECIS)*. AIS, 2020.

[10] R. Bose, R. Mans, and W. van der Aalst, "Wanna improve process mining results?" in *IEEE CIDM*. IEEE, 2013, pp. 127–134.

[11] S. Suriadi, R. Andrews, A. ter Hofstede, and M. Wynn, "Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs," *Information Systems*, vol. 64, pp. 132–150, 2017.

[12] W. van der Aalst, "Process mining: Discovering and improving spaghetti and lasagna processes," in *IEEE CIDM, 2011*. IEEE, 2011, pp. 1–7.

[13] R. Andrews, S. Suriadi, C. Ouyang, and E. Poppe, "Towards event log querying for data quality," in *OTM Confederated Int. Confs "On the Move to Meaningful Internet Systems"*. Springer, 2018, pp. 116–134.

[14] R. Andrews, C. van Dun, M. Wynn, W. Kratsch, M. Röglinger, and A. ter Hofstede, "Quality-informed semi-automated event log generation for process mining," *Decis. Support Syst.*, p. 113265, 2020.

[15] F. Fox, V. Aggarwal, H. Whelton, and O. Johnson, "A data quality framework for process mining of electronic health record data," in *IEEE Int. Conf. on Healthcare Informatics (ICHI)*. IEEE, 2018, pp. 12–21.

[16] N. Weiskopf, S. Bakken, G. Hripcsak, and C. Weng, "A data quality assessment guideline for electronic health record data reuse," *Egems*, vol. 5, no. 1, 2017.

[17] D. Savan, *An Introduction to CS Peirce's Full System of Semeiotic*. Toronto Semiotic Circle, 1988.

[18] M. Marshall, "Sampling for qualitative research," *Family practice*, vol. 13, no. 6, pp. 522–526, 1996.

[19] M. Tongco, "Purposive sampling as a tool for informant selection," *Ethnobotany Research and applications*, vol. 5, pp. 147–158, 2007.

[20] R. Longhurst, "Semi-structured interviews and focus groups," *Key methods in geography*, vol. 3, no. 2, pp. 143–156, 2003.

Queueing Inference for Process Performance Analysis with Missing Life-Cycle Data

Guy Berkenstadt*, Avigdor Gal*, Arik Senderovich†, Roee Shraga* and Matthias Weidlich‡
*Technion – Israel Institute of Technology, †University of Toronto, ‡Humboldt-Universität zu Berlin
{guy.ber@campus.,avigal@}technion.ac.il, arik.senderovich@utoronto.ca,
shraga89@campus.technion.ac.il, matthias.weidlich@hu-berlin.de

***Abstract*—Measuring key performance indicators, such as queue lengths and waiting times, using event logs serve for improvement of resource-driven business processes. However, existing techniques assume the availability of complete life cycle information, including the time a case was scheduled for execution (aka arrival times). Yet, in practice, such information may be missing for a large portion of the recorded cases. In this paper, we propose a methodology to address missing life-cycle data by incorporating predicted information in business processes performance analysis. Our approach builds upon techniques from queueing theory and leverages supervised learning to accurately predict performance indicators based on an event log with missing data. Our experimental results using both synthetic and real-world data demonstrate the effectiveness of our approach.**

I. Introduction

Performance analysis is an important task when improving business processes [1]. Accurate and robust estimation of performance measures, such as cycle times and queue-lengths, is crucial for locating bottlenecks and understanding system dynamics [2]. In heavily loaded processes where cases compete over scarce resources, the use of queue mining, which is the discovery of the queueing perspective in process mining [3], plays a key role in performance analytics [4].

Common queue mining methods assume that one observes a complete activity life cycle, including arrival times (when an activity is scheduled), activity start times, and completion times. However, arrival times are often not observed in the data [5], in particular for processes in which a manual service is provided, such as patient treatments in hospitals. The reason being that data is often recorded only for the service providers, not the requesters. For instance, the start and end of a treatment step for a patient is recorded in a health information system, whereas the patient's arrival in the waiting room for the treatment is not. Attempts to obtain the missing data may involve incentives for self-reporting by service requesters or explicit sample-based observation. Yet, arrival times will, in general, only be available for a small portion of the cases.

In this work, we target performance analysis of business processes without assuming fully observable activity life cycles. Specifically, in our setting, the time that an activity was scheduled for execution, aka the arrival time, is assumed to be only partially available, *e.g.,* through self-reporting or sampling. Hence, queueing measures that were shown to be of high value when analyzing a process' performance, see [6], cannot be computed exactly based on historical data.

The problem of estimating the queueing features in historical data was studied in queueing theory in the past [7]. A statistical method that provides performance estimation (*e.g.,* queue length distribution) under missing data is referred to as a *queueing inference engine* (QIE) [7]–[10]. QIEs make use of various queueing assumptions, *e.g.,* Poisson arrivals and first-come first-served (FCFS) dispatch policies, to provide an accurate distribution over queueing measures in single-station systems [7] and queueing networks [10].

Taking up these results, one may use a QIE to infer queue lengths and other congestion features in hindsight. However, as many analytical approaches in queueing theory, a QIE suffers from the curse of simplicity, *i.e.,* unrealistic and overly restrictive assumptions. Arrivals in real processes may not be Poisson, but depend on deterministic appointment times. Also, cases may be classified into types, which show differences in service times. Lastly, dispatch policies may not adhere to the order of arrivals, thus violating the FCFS assumption.

The contribution of this paper is a methodology to overcome this curse of simplicity. We propose to utilize a QIE to generate improved predicted values of performance indicators through supervised learning, thereby mitigating the modelling biases imposed by a QIE's assumptions. To this end, we combine features created from a QIE with temporal features directly extracted from recorded data to correct inherent estimation errors of the QIE. Features of both types are then used to learn a reconstruction of queue lengths and waiting times.

We perform a thorough empirical evaluation of the approach. Using a synthetic dataset that satisfies QIE assumptions, we show that while QIE-based features perform well, additional features derived from the log may assist in reducing prediction error. We also experimented with real-world data from an outpatient cancer hospital in the United States. Here, QIE assumptions no longer hold and as a result, relying solely on QIE-based features offer an inferior solution to a combined set of features that uses both QIE-based and additional log-based features generate improved approximations for the measures of interest (*e.g.,* we observe a 48% improvement of queue length prediction over the baseline).

The rest of the paper is organized as follows. Section II introduces our model and problem definition, along with background on queueing theory. Our proposed solution is given in Section III. Section IV details our evaluation, followed by related work (Section V) and concluding remarks (Section VI).

978-1-7281-9832-3/20/$31.00 ©2020 IEEE
DOI 10.1109/ICPM49681.2020.00019

TABLE I: Example log including life cycle information.

Case	Activity	Life Cycle State	Timestamp	Resource
11	Vitals	start	6/7/20 10:30	Yorke
12	Consultancy	start	6/7/20 10:38	Greenwood
11	Vitals	end	6/7/20 10:45	Yorke
13	Vitals	start	6/7/20 10:55	Yorke
11	Imaging	start	6/7/20 11:05	O'Brien
12	Consultancy	end	6/7/20 11:10	Greenwood
13	Vitals	end	6/7/20 13:50	Yorke
13	Consultancy	start	6/7/20 14:05	Greenwood
11	Imaging	end	6/7/20 14:10	O'Brien
13	Consultancy	end	6/7/20 14:15	Greenwood

II. Model and Problem Definition

We start by presenting a standard event log model that shall be used throughout the paper. Next, we define two process performance indicators (PPIs), namely queue length and waiting time, and formulate the problem of their prediction. We conclude the section with some background on the queueing inference engine (QIE), a set of methods used for estimating PPIs from missing event data.

A. Event Data with Life Cycle Information

Let $\mathcal{E}$ be the universe of events. We assume that an event $e \in \mathcal{E}$ is a quintuple, $e = (i, a, s, t, r)$ such that,

- $i \in \mathcal{I}$ is a unique case identifier,
- $a \in \mathcal{A}$ is an activity,
- $s \in \{start, end\}$ is a life cycle state denoting start or completion of activity execution,
- $t \in \mathcal{T}$ is a timestamp associated with the event,
- $r \in \mathcal{R}$ is the resource that executes the activity.

Let $\mathcal{E}^*$ be the set of all finite sequences over $\mathcal{E}$. Then, a case i corresponds to a trace of events, $\sigma_i \in \mathcal{E}^*$ of length n_i. We denote by $\sigma_{i,j}$ the j-th event in trace σ_i with $j = 1, \ldots, n_i$. We refer to the quintuple of attributes of event $\sigma_{i,j}$ by $(i, a_{i,j}, s_{i,j}, t_{i,j}, r_{i,j})$. Moreover, we assume that the information on life cycles of activity executions per case is complete and unambiguous, *i.e.,* there are matching *start* and *end* state transitions for all activity executions and multiple executions of the same activity do not overlap within a trace. Formally, for an event with $s_{i,j} = start$ ($s_{i,j} = end$), there is a later (earlier) event, $j < l$ ($j > l$), with $a_{i,j} = a_{i,l}$ and $s_{i,l} = end$ ($s_{i,l} = start$), and for all events in between, $j < k < l$ ($l < k < j$), it holds that $a_{i,k} \neq a_{i,j}$.

An event log $\mathcal{L} \subseteq \mathcal{E}^*$ is defined as a set of traces. The event log covers a time period $[t_{min}, t_{max}]$ with t_{min} (t_{max}) being the minimal (maximal) timestamp observed in the log.

Table I illustrates our model of an event log. It contains events about treatment steps, which are either *vitals*, *consultancy*, or *imaging*, of three patients in a hospital. Events indicate the start and completion of the respective activities.

We assume that there is a one-to-one correspondence between an activity and a resource, and hence, the underlying system can be described as a system of queues with each queue containing cases that wait to be processed by the corresponding resource. However, we do not enforce any assumption in terms of a single entrance into the process. That is, each of the activities and, hence queues, may experience exogenous as well as internal arrivals.

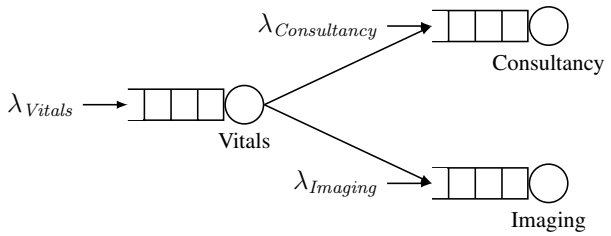


Fig. 1: System of queues for the activities of the example log.

Fig. 1 illustrates a system of queues for the aforementioned example log. The process defines that *vitals*, if conducted, happen before *consultancy* and *imaging*. Yet, some patients arrive directly to these treatment steps.

In this work, we shall assume that information about external arrivals is missing. For internal arrivals, we consider the transition time to be negligible, *i.e.,* the arrival time corresponds to the completion time of the previous activity. More specifically, given a case with trace σ_i, the only unknown timestamp is that of the first event in a trace, namely $\sigma_{i,1}$. This event may relate to one out of many possible activities to start a process instance, though.

B. Predicting Process Performance Indicators

In resource-driven systems, cases compete over scarce resources and form queues. Hence, we consider two key PPIs for measuring performance of such systems, namely queue length and waiting time (or delay) [11]. The two PPIs reflect the congestion level of the system and are known to be strongly correlated with client satisfaction [12], quality of service [13] and resource utilization [11]. For example, knowing the queue length per resource over time may help in identifying bottleneck resources in the process [14].

Typically, the PPIs are considered over time (*i.e.,* presented as a time series) and their distribution at different points in time is of high interest. Formally, we denote by $\mathcal{Q}_r(t) \geq 0, t \geq 0$ the queue length stochastic process that may be realized (computed from the data) with non negative values $q_r(t) \geq 0$. For example, by writing $q_1(10) = 5$ we mean that the first resource queue had the value of 5 at the 10th minute (assuming minute time units). Clearly, high values of observed queue lengths correlate with busyness of the system and congestion.

Another measure of interest is the stochastic process that represents the (virtual) waiting time for a case that arrives at time t (denoted $\mathcal{W}_r(t), t \geq 0$). Several works link $\mathcal{W}_r(t)$ with customer dissatisfaction and churn [15], [16]. The realization of $\mathcal{W}_r(t)$ is denoted by $w_r(t)$.

When executing a business process, we wish to compute $\hat{q}_r(t)$ and $\hat{w}_r(t)$, prediction of the true values of the realizations of the two stochastic processes ($q_r(t)$ and $w_r(t)$) as they were observed over the time period of the event log ($[t_{min}, t_{max}]$). Throughout this work, we shall assume full independence between the different resources. Thus, our methods will consider every resource in the queueing network to be a single-station queue for which we wish to predict the actual queue length

and virtual waiting over the recorded time period. Hence, we drop the r subscript from the stochastic processes (and their realizations) and write $\mathcal{Q}(t)$ and $\mathcal{W}(t)$ (with realizations $q(t)$ and $w(t)$), respectively.

We are now ready to define the queue length and waiting time prediction problem.

Problem 1 (Queue Length and Waiting Time Prediction)**.** *Let $q(t)$ and $w(t)$ be the realizations of queue length and virtual waiting times at time t, respectively. Given an event log $\mathcal{L}$ over time period $[t_{min}, t_{max}]$, the Queue Length and Waiting Time Prediction problem is to provide accurate predictors $\hat{q}(t)$ and $\hat{w}(t)$, for all $t \in [t_{min}, t_{max}]$.*

State-of-the-art techniques from statistical queueing theory construct exact expressions for the distribution of $\mathcal{Q}(t)$ at every time point t by making simplifying assumptions regarding the underlying system. Once the distribution is obtained, one may predict the missing queue lengths by, *e.g.,* using the mean values of the distribution. These methods are collectively referred to as the queueing inference engine (QIE) [7], [8], [10]. Specifically, QIE methods receive missing data $\mathcal{L}$, in which only start and completion times of activity executions are observed, and provide an exact distribution for the queue length ($\pi_{\mathcal{L}}(\mathcal{Q}(t))$) for every point in time $t \in [t_{min}, t_{max}]$:

$$\pi_{\mathcal{L}}(\mathcal{Q}(t)) = P(\mathcal{Q}(t) = q(t) \mid \mathcal{L}) \tag{1}$$

where $P(X = x \mid Y = y)$ is the distribution of X conditioned on $Y = y$. We write the conditioning on the event log, meaning that we condition on all available information in the log.

However, when trying to solve Problem 1, there are three main issues when applying the QIE. First, it enforces various assumptions, *i.e.,* Poisson arrivals, a first-come first-served (FCFS) service policy, and non-idling resources. These assumptions may not hold true in real-world systems. Second, it provides the distribution of the waiting times for each individual case. Yet, it does not deliver a distribution over $\mathcal{W}(t)$ that would enable the construction of an estimator for $\hat{w}(t)$, *i.e.,* for the waiting time for an arbitrary time point. Third, the QIE neglects additional contextual information that may appear in the log, beyond start and completion times.

In this work, we show how to overcome these limitations. To this end, we will embed the QIE in a methodology for supervised learning to predict both, $q(t)$ and $w(t)$, without imposing strong assumptions on the investigated system.

III. Queueing Inference with Supervised Learning

We next present an approach for solving Problem 1 (Section II). The approach uses supervised learning based on some event data with full life cycle information, *i.e.,* data that contains information on external arrivals that do not appear in the original log $\mathcal{L}$. Formally, we assume the existence of a (commonly partial) measurement function,

$$\psi : \mathcal{I} \nrightarrow \mathcal{A} \times \{arrival\} \times \mathcal{T} \times \mathcal{R},$$

that maps some cases to exogenous arrival events. In essence, it maps a case identifier to the corresponding event with *arrival* being this event's life cycle state.

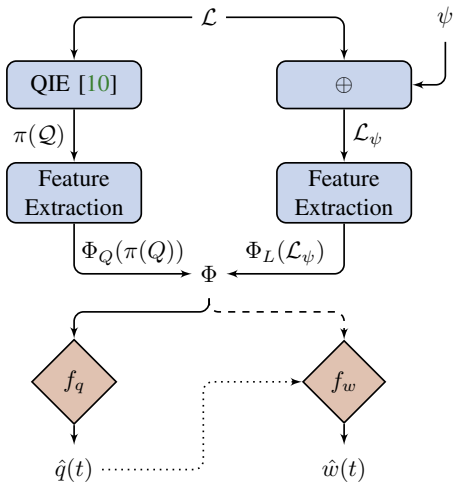


Fig. 2: Approach to queue length and waiting time prediction.

For example, ψ can be obtained by providing incentives for self-reporting, by purchasing observation services, or by placing dedicated sensors to measure the arrivals for some cases. Given ψ with $\mathcal{I}' \subset \mathcal{I}$ as its domain, the training log is denoted $\mathcal{L}_\psi$ such that,

$$\mathcal{L}_\psi = \mathcal{L} \cup \{i \oplus \psi(i) \mid i \in \mathcal{I}'\},$$

with $\oplus$ being a concatenation operator. Equipped with a training log, we turn to describe our technique for predicting missing queue lengths and virtual waiting times.

An overview of the approach is illustrated in Fig. 2. We start by solving the problem of inferring missing queue lengths. We do so by creating a feature representation Φ and learning a supervised model based on the training log $\mathcal{L}_\psi$. After predicting the queue length values $\hat{q}(t)$, we may proceed to predict $\hat{w}(t)$ as a function of $\hat{q}(t)$ (dotted arrow in Fig. 2), using $\hat{q}(t)$ as a derived feature. Alternatively, we can use the same feature representation Φ to learn a supervised model, thus directly predicting $\hat{w}(t)$ (dashed arrow in Fig. 2).

Starting from the event log, we use two sources to generate the feature set. The first source is the QIE approach of constructing queue length distribution conditioned on missing data [10]. This results in the exact queue length distribution $\pi(\mathcal{Q})$[1] under a set of queueing assumptions (top left of Fig. 2).

From this distribution, we extract a feature set $\Phi_Q(\pi(\mathcal{Q}))$, to be described in Section III-A1. The second source we use is the training log $\mathcal{L}_\psi$, i.e. the original log enriched with arrival measurements, from which process-aware features $\Phi_L(\mathcal{L}_\psi)$ are extracted (Section III-A2). The two feature sets are combined to create a feature representation $\Phi = \Phi_Q(\pi(\mathcal{Q})) \cup \Phi_L(\mathcal{L}_\psi)$, which is used to predict the actual historical queue length (Section III-B) and the virtual waiting time at each point in time (Section III-C).

A. Designing the Feature Space

The queueing inference engine enables us to obtain $\pi(\mathcal{Q})$, which is the exact theoretical queue length distribution, condi-

[1]We drop the time index t for brevity reasons.

tioned on $\mathcal{L}$, under the aforementioned assumptions [10]. The mean of the distribution $\pi(\mathcal{Q})$ may then serve as a prediction for the queue length, as the expected value (conditional on data) is known to be the best least-squares predictor of the value itself [17].

Yet, as mentioned in Section II-B, applying the QIE in our context means that (i) simplistic modelling assumptions are imposed; (ii) analysis is limited to the waiting time distribution per case, rather than for an arbitrary point in time; and (iii) contextual information in the event log is neglected. Against this background, we propose a feature encoding function Φ_Q over the queue distribution provided by the QIE, which maps an estimated queue distribution $\pi(\mathcal{Q})$ to a feature vector (Section III-A1). In addition, we extract contextual features from the event log (Section III-A2). All these features serve us in predicting the actual queue length (Section III-B).

1) Feature Extraction from the Queue Distribution: The first novel component in our framework is the extraction of features from a given queue distribution $\Phi_Q(\pi(\mathcal{Q}))$.

We partition the feature set into three groups, each dedicated to a different property of the distribution we aim to capture. In what follows, we let q_{max} be a value such $P(\mathcal{Q} > q_{max}) \simeq 0$, *i.e.,* we assume that the distribution is truncated at q_{max}.

Measures of Central Tendency: We consider the mean, median, and mode as three main statistical quantities (that are identical that for symmetric distributions). The mean (aka first moment) is defined in our context as:

$$\phi_M = M = \sum_{q=0}^{q_{max}} q \cdot P(\mathcal{Q} = q) \tag{2}$$

Since the mean measure is particularly susceptible to the influence of outliers and is not well-suited to identify a central point in skewed distributions, we also consider the median. For our setting, it is computed as follows:

$$\phi_{median} = \underset{q \in [0, q_{max}]}{\operatorname{argmin}} \; P(\mathcal{Q} \leq q) \geq \frac{1}{2} \tag{3}$$

Finally, to be able to cope with nominal scales, we incorporate the mode, *i.e.,* the most common value of the distribution:

$$\phi_{mode} = \underset{q \in [0, q_{max}]}{\operatorname{argmax}} \; P(\mathcal{Q} = q) \tag{4}$$

We use all three measures in $\Phi_Q(\pi(\mathcal{Q}))$. Note that $\Phi_Q(\pi(\mathcal{Q}))$ is not complete and further measures for central tendency, *e.g.,* the geometric mean, may be employed.

Distribution Quantiles: To encode more information from the distribution, we discretize it using the quantiles of $\pi(\mathcal{Q})$. Specifically, we incorporate the deciles of the distribution, *i.e.,* 10-quantile. The i-th decile is given by:

$$\phi_{qi} = \underset{q \in [0, q_{max}]}{\operatorname{argmin}} \; P(\mathcal{Q} \leq q) \geq \frac{i}{10} \tag{5}$$

where the 5-th decile is the distribution's median.

Distribution Dispersion: We further capture the dispersion of the distribution by its standard deviation (referred to as second moment in its squared form):

$$\phi_{sd} = sd = \sqrt{\sum_{q \in [0, q_{max}]} (q - M)^2 \cdot P(\mathcal{Q} = q)} \tag{6}$$

Lower σ values indicate that most values tend to be close to the mean, while higher σ indicate dispersion over a wider range. We next suggest two additional (less common) moments of a distribution to serve as additional predictors for dispersion.

The third moment, skewness, measures the asymmetry of the probability distribution with respect to its mean.

$$\phi_\gamma = \sum_{q \in [0, q_{max}]} (\frac{q - M}{sd})^3 \cdot P(\mathcal{Q} = q) \tag{7}$$

Unlike standard deviation, skewness can also be negative, indicating that a significant part of the distribution is concentrated on smaller values and the mean values of the distribution shift left to the center. Positive skewness means that the distribution is right-tailed and the mean tends to the right as well.

Finally, we use the fourth moment kurtosis to measure the tailedness of the probability distribution.

$$\phi_\kappa = \sum_{q \in [0, q_{max}]} (\frac{q - M}{sd})^4 \cdot P(\mathcal{Q} = q) \tag{8}$$

Similar to skewness, kurtosis describes the shape of a probability distribution and is especially useful to quantify the the effect of outliers. Larger kurtosis indicates a significant influence of outliers on the distribution.

2) Feature Extraction from the Event Log: We extract features from the event log, $\mathcal{L}$, focusing on log-based features at a specific time t, for which we aim to predict the queue length, $\hat{q}(t)$. Specifically, we use the elapsed time in the period ($\phi_t = t - t_{min}$) and the number of observed timestamps prior to the current one ϕ_n, defined as follows. Let

$$a_t = \max_{e_i \in \mathcal{L}} \{t_i \mid s_i = arrival \wedge t_i \in [t_{min}, t]\},$$

then, ϕ_n can be written as,

$$\phi_n = |\{e_i \in \mathcal{L} : s_i = start \wedge t_i \in [t_{min}, a_t]\}|.$$

B. Predicting the Queue Length

Having obtained a combined feature set $\Phi = \Phi_Q(\pi(\mathcal{Q})) \cup \Phi_L(\mathcal{L})$, we aim to find a queue length estimator. To this end, we cast our problem as a regression problem aiming to find regressor f_q. To implement this regressor we use an out-of-the-box approach, experimenting with several types of regression models including linear regression and gradient boosting.

For training we assume the availability of a labeled set containing respective labels $q(t)$ and $w(t)$ for $t \in [t_{min}, t_{max}]$.

In test time, given an event log $\mathcal{L}$, we use the QIE to produce $\pi(\mathcal{Q})$, from which we extract the feature sets $\Phi_L(\mathcal{L})$ and $\Phi_Q(\pi(\mathcal{Q}))$, respectively. Then, we apply f_q to estimate the queue length $\hat{q}(t)$ for every observed time $t \in [t_{min}, t_{max}]$.

C. Predicting Virtual Waiting Times

Having predicted $\hat{q}(t)$, we now turn our efforts to the prediction of the (virtual) waiting times. Similar to queue length prediction, we cast the waiting time estimation as a regression problem and aim to find a regressor f_w that predicts $\hat{w}(t)$. We present two alternatives to learn function f_w.

1) Using the Predicted Queue Length: The first alternative we consider uses the estimation of $\hat{q}(t)$ as a single feature. Specifically, $M/M/1$ queues satisfy the following three assumptions: (1) arrival is according to a Poisson process, (2) service times are exponentially distributed with some mean $\frac{1}{\mu}$ and are independent of each other and of inter-arrival times, and (3) the dispatching policy is First-Come-First-Served. These assumptions lead to the following best predictor [4]:

$$\hat{w}(t) = (\hat{q}(t) + 1)/\mu \tag{9}$$

Therefore, using $\hat{q}(t)$ as the only feature in a linear regression model, should yield (under the relevant queueing assumption) $\frac{1}{\mu}$ as the coefficient of $\hat{q}(t)$ and $\frac{1}{\mu}$ as the intercept. However, since the underlying data does not necessarily adhere to the $M/M/1$ model, we simply feed $\hat{q}(t)$ as a single feature into f_w. We consider both linear and non-linear models of f_w, testing the robustness of our approach with respect to various choices of the machine learning algorithm.

2) Using the Feature Set: An alternative solution directly uses Φ to learn a regressor f_w, estimating $\hat{w}(t)$ eliminating $\hat{q}(t)$ as a feature. Note that f_w and f_q are different mappings, using the same feature set but trained using different labels based on $\mathcal{L}_\psi$ (using $w(t)$ and $q(t)$, respectively).

IV. Evaluation

In this section we evaluate our methodology using datasets, as detailed in Section IV-A. Our experimental setup is given in Section IV-B. Section IV-C reports our main results.

A. Datasets

We rely on two datasets, a synthetic log coming from an $M/M/1$ queueing system simulator, and a real-world dataset coming from the Dana-Farber Cancer Institute (DFCI), a large outpatient hospital in Boston, USA.

MM1: To test our method in a controlled setting, we created a synthetic event log using an $M/M/1$ simulator. It produces timestamped arrival, start, and completion events. We used five different values for the arrival rate ($\lambda \in \{.84, .86, .88, .9, .92\}$) and a constant service rate of $\mu = 1$ (a common scenario [18]) for the simulations. For each λ value, we sampled $55,000$ inter-arrival times (for a total of 196,459 cases) and recorded their execution to create the event log.

DFCI: We consider a real-world dataset of the Dana-Farber Cancer Institute, a large outpatient cancer hospital in the US. It originates from a process that contains an appointment schedule and two types of patients. The log contains $154,777$ cases over two and a half years between 2014 and 2016 coming from the blood draw station. The average queue length is 15.90 and the average virtual waiting time is 20.53 minutes. The DFCI log provides two additional elements: (1) a schedule, which is used to measure the delay between the time of prediction and the respective time ($delay$), and (2) customer type ($type$) that represents the complexity level of the patient. To demonstrate the usefulness of this information that is often available in real-life logs we add the two features to Φ (dubbed Φ^+).

For the annotation and prediction of $w(t)$ we use the waiting time of the last customer that arrived at the system at time t.

B. Experimental setup

Our method was implemented in Python and is publicly available.[2] We used scikit-learn[3] for supervised learning.

1) Evaluation Measures: The prediction quality of queue length $\hat{q}$ and waiting time $\hat{w}$ is evaluated using 1) the root mean squared error (RMSE), 2) the mean absolute error (MAE), and 3) the absolute relative error (MARE).

Let $y_1, y_2, \ldots, y_k$ and $\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_k$ be a set of k real values and the corresponding predictions of a PPI y (queue length or waiting time), respectively.

RMSE measures the prediction accuracy and is given by:

$$RMSE = \sqrt{\frac{1}{k}\sum_{i=1}^{k}(y_i - \hat{y}_i)^2}, \tag{10}$$

where low RMSE indicates that the predictor is precise.

The next two measures are MAE and MARE, which are less sensitive to outliers compared to the RMSE, and hence worth exploring. They are defined as follows,

$$MAE = \frac{1}{k}\sum_{i=1}^{k}|y_i - \hat{y}_i|, MARE = \frac{1}{k}\sum_{i=1}^{k}\frac{|y_i - \hat{y}_i|}{y_i}. \tag{11}$$

MAE measures the absolute error and MARE normalizes it with respect to the real values. Whenever MARE is undefined (*i.e.,* $y_i = 0$), its value is omitted from the computation. Note that these values are rare.

2) Experimental Methodology: For each dataset our evaluation is randomly partitioned into two parts, namely, a training set (75%) and a test set (25%).

During training, we apply supervised learning over the training set to produce $f_q(\Phi)$ (Section III-B) and $f_w(\Phi)$ (Section III-C2). We report results of linear and gradient boosting [19] regressors, selected empirically from a set of regressors including, among others, XGBoost and Random Forest. Once $f_q(\Phi)$ is trained, we use its predictions over the training set to train a single feature model $f_w(\hat{q})$ (Section III-C1). We also train f_q and f_w on subsets of the features.

For testing, we use $f_q(\Phi)$ to predict queue lengths and $f_w(\Phi)$ and $f_w(\hat{q})$ to predict waiting times over the test set.

C. Results

We start with a comparative analysis of average performance (Section IV-C1), followed by an illustrative qualitative analysis (Section IV-C2), and a feature analysis (Section IV-C3).

[2] https://github.com/guy-ber/Queueing-Inference-with-Missing-Data.git
[3] https://scikit-learn.org

TABLE II: Comparison: Baseline vs. our approach.

		MM1			DFCI		
Queue Length Prediction		RMSE	MAE	MARE	RMSE	MAE	MARE
Baseline	ϕ_M	4.690	**2.903**	**0.434**	9.717	7.308	0.480
Linear Regression	$f_q(\Phi_Q(\pi(\mathcal{Q})))$	5.094	3.077	0.467	7.898	6.216	0.631
	$f_q(\Phi)$	5.252	3.145	0.467	7.636	5.995	0.600
	$f_q(\Phi^+)$	-	-	-	7.433	5.834	0.584
Gradient Boosting	$f_q(\Phi_Q(\pi(\mathcal{Q})))$	5.082	3.063	0.467	7.464	5.807	0.546
	$f_q(\Phi)$	**4.675**	2.917	0.461	6.873	5.524	0.488
	$f_q(\Phi^+)$	-	-	-	**6.554**	**4.996**	**0.461**
Waiting Time Prediction		RMSE	MAE	MARE	RMSE	MAE	MARE
Baseline	$(\phi_M+1)/\mu$	**5.774**	4.037	2.762	-	-	-
Linear Regression	$f_w(\hat{q})$	5.921	3.911	1.741	17.553	12.840	2.422
	$f_w(\Phi_Q(\pi(\mathcal{Q})))$	6.235	3.913	1.882	17.290	12.652	2.280
	$f_w(\Phi)$	6.351	3.929	1.857	17.173	12.553	2.292
	$f_w(\Phi^+)$	-	-	-	17.024	12.418	2.278
Gradient Boosting	$f_w(\hat{q})$	6.344	4.014	1.917	17.433	12.758	2.348
	$f_w(\Phi_Q(\pi(\mathcal{Q})))$	5.787	**3.623**	**1.599**	17.141	12.475	2.257
	$f_w(\Phi)$	5.860	3.638	1.611	16.722	12.052	2.220
	$f_w(\Phi^+)$	-	-	-	**16.519**	**11.885**	**2.191**

1) Average Performance: Table II reports on the average performance of our approach over the two datasets, using RMSE, MAE, and MARE as evaluation measures. At the top, we present the performance of three queue length predictors: a baseline model (ϕ_M, see Section III-A), a model that uses the distribution features only ($f_q(\Phi_Q)$, Section III-A1), and a model using all features, ($f_q(\Phi)$), including log-based features (Section III-A2). For the last two, we present results using linear and gradient boosting regressors. The lower part lists the waiting time prediction with Eq. 9 as a baseline method, a model trained over the distribution features only ($f_w(\Phi_Q)$), a model trained using all the features ($f_w(\Phi)$), and a model trained over a the predicted queue length ($f_w(\hat{q})$). The formula $(\phi_M+1)/\mu$ cannot be applied for the DFCI dataset since it violates the single-server assumption. For the DFCI dataset, we further consider $f_q(\Phi^+)$ and $f_w(\Phi^+)$, which include the additional features $delay$ and $type$ (see Section IV-A).

Our empirical analysis indicates that gradient boosting outperforms linear regression, probably due to its ability to capture non-linear relationships between features. Therefore, we focus hereinafter on the results of gradient boosting regression.

The MM1 dataset presents a sterile setting, for which using ϕ_M should suffice to achieve the best prediction for $q(t)$ [4]. While this holds for MAE and MARE, we see that using Φ does provide a slight improvement in RMSE. A reason may be the presence of outliers, which are better handled by ϕ_M. For waiting time prediction, the baseline provides the best RMSE, while a distribution-based model achieves the best MAE and MARE. Hence, when analyzing an $M/M/1$ queuing system, non-queue-based features are superfluous.

For the DFCI dataset, the full feature set (Φ^+) provides the best prediction of both queue length and waiting time, compared to features coming from both the log ($\Phi_{\mathcal{L}}$) and the QIE ($\Phi_{\mathcal{Q}}$). Specifically, the additional features ($delay$ and $type$) yield an improvement. For instance, when using $f_q(\Phi^+)$ we get a 48% RMSE increase over the baseline. This demonstrates the effectiveness of our approach in real-world scenarios, especially, when additional features are available.

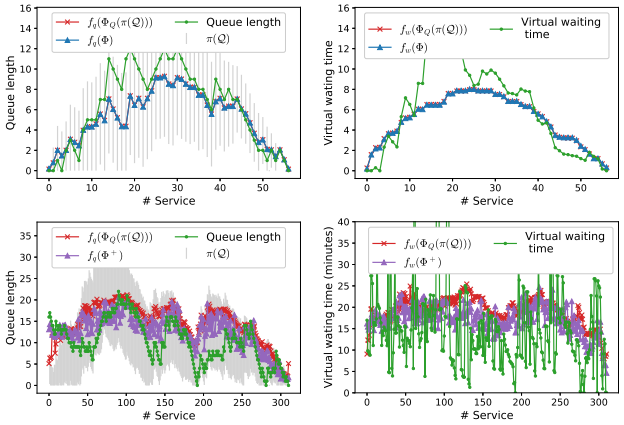


Fig. 3: Queue length (left) and waiting time (right) prediction over an example time period, MM1 (top) and DFCI (bottom)

2) Qualitative Analysis: To explore the results of Section IV-C1, we conduct a qualitative analysis of two time periods from the experiments (Fig. 3).

The shaded area in the queue length prediction figures represents $\pi(\mathcal{Q})$ (Eq. 1), showing the area covered by 1.96 standard deviation from the mean of the distribution (and censored from below at 0). The distribution represents an aggregated view of all likely scenarios of the queue length over time. As such, it is only natural that the results for the MM1 dataset (top left of Fig. 3) is mostly within the shaded area, although it shows a certain variability over time. The figure also presents the predictions of $f_q(\Phi_Q(\pi(\mathcal{Q})))$, in red, with features extracted from the queue distribution, and using the full set of features $f_q(\Phi)$ in blue. We can see that both models yield similar results, with the ability to rise and fall with the actual behavior of the instance, yet not venturing too far away from the center of the shaded area.

A different picture is revealed for the real-world data where we present the full feature set $f_q(\Phi^+)$ that includes the two additional features of $delay$ and $type$ (denoted in purple). While the instance remains mainly within the distribution shaded area, we observe a deviation at the very beginning of the tested interval, which is likely due to violation in the arrival time assumptions. Initially, patients likely arrived in a bulk rather than according to a Poisson process as the QIE assumes. Here, we also see different performance between the models. While $f_q(\Phi_Q(\pi(\mathcal{Q})))$, in red, fails to identify the deviation at the beginning, $f_q(\Phi^+)$, in purple, adapts the prediction to fit the peculiarities of the specific instance, presumably utilizing the features extracted from the log.

The prediction of waiting time shows a similar pattern. For the MM1 dataset, both models exhibit a similar behavior, rising and falling with the actual waiting time, yet remaining cautious, which hurts the prediction whenever it goes to the extreme. For the real-world data, using the full set of features yields a more agile model that identifies better the changes of the actual waiting time. In particular, a model that uses the distribution features alone yields higher overestimation in the middle of the interval than a model using all available features.

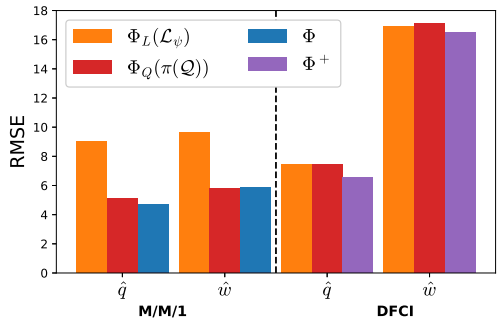


Fig. 4: RMSE for models trained using the distribution-based features only ($\Phi_Q(\pi(\mathcal{Q}))$), the log-based features only ($\Phi_L(\mathcal{L}_\psi)$), and all features (Φ for MM1 and Φ^+ for DFCI).

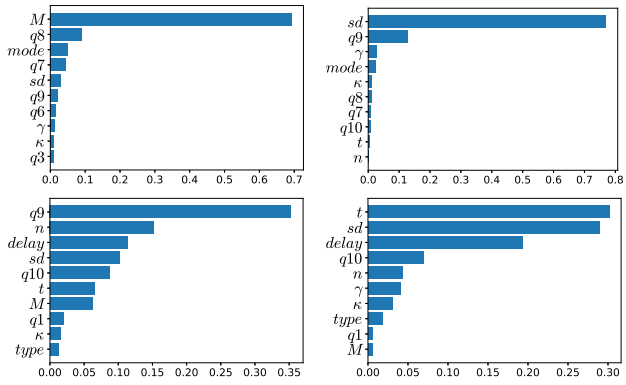


Fig. 5: Queue length (left) and waiting time (right) feature importance over the MM1 (top) and DFCI (bottom) datasets

3) Feature Analysis: Next, we assess the effect of features on prediction quality. First, we compare the results of the two groups of features ($\Phi_Q(\pi(\mathcal{Q}))$ and $\Phi_L(\mathcal{L}_\psi)$) with the full feature set, namely Φ for MM1 and Φ^+ for DFCI. Then, we analyze the individual features impact using Gini importance.

Fig. 4 demonstrates the varying feature set importance between settings. For MM1, log-based features are insufficient to accurately predict both $q(t)$ and $w(t)$. Yet, for queue length prediction, its addition to distribution-based features improves the results of $f_q(\Phi_Q(\pi(\mathcal{Q})))$. For waiting time prediction, the distribution-based features suffice as the waiting time in an $M/M/1$ queue is exponential (and thus memoryless) in steady-state. The DFCI dataset tells a different story where the distribution-based and log-based features achieve competitive results yet the full model takes the best of both worlds.

Finally, we explore the impact of individual features. As expected, for MM1, the distribution mean is the most dominant feature for queue length prediction (top left of Fig. 5). For the waiting time prediction (top right of Fig. 5), the distribution dispersion features are dominant. Fig. 4 further suggests that log-based features are ineffective for the MM1 dataset. For DFCI, features such as $delay$, n, and t influence both predictions. The upper quantiles ($q9$ and $q8$) are also influential, capturing the right tail of the distribution.

V. Related Work

Prediction of performance properties of a process using event logs has been approached from various angles [20]. A first class of techniques is generative. Performance characteristics are assumed to be tied to some underlying process, so that control-flow discovery is conducted first, before enriching the model with performance characteristics, *e.g.,* [21]–[25]. Then, analytical or simulation-based time prediction is facilitated.

Other techniques are discriminative and leverage machine learning algorithms, mostly for regression, *e.g.,* XGBoost and neural networks. Features that capture control-flow information, but also data linked to cases, or context information are used as a starting point [26]–[28]. Yet, the focus is on intra-case features, neglecting interferences among cases.

For resource-driven processes, it has been recognized that performance analysis requires the inclusion of congestion levels. Queueing effects may be considered for time prediction in generative methods by choosing an appropriate target formalisms, such as Generalized Stochastic Petri nets (GSPNs) [29] or queueing networks [30]. Discovery of these models adopts traditional control-flow discovery for the structure and extends it with estimations of model parameters [2], such as arrival rates or resource capacities.

For discriminative approaches, queueing features are often designed manually, *e.g.,* the Q-Lasso method [31], proposed for waiting time prediction in healthcare. Such a manual approach is time-consuming and requires specialized expertise.

Techniques for automated feature generation, see [32] for a review, try to avoid the above limitations by using generic functions for feature transformation (*e.g.,* sine, squared root, log). Useful transformations are constructed using local search [33], reinforcement learning [32], and deep learning [34]. While being generally applicable, the main drawbacks of these methods are their computational complexity along with their limited ability to derive complex features.

Focusing on process performance analysis, a process may be approximated by a single-station queue to derive predictive features [4]. Beyond single-station models, discovery of congestion graphs enables the construction of dynamic queueing features [35]. In [36], the authors construct inter-case features including a count of the number of cases and inter-event durations for cases that share similar prefixes. In a queueing system, these features translate to the number of customers in queue and in service, and waiting and service times. Yet, in [4], [35], [36], full life cycle information is assumed, which renders the approaches inapplicable in our setting.

Discovering queues from event logs with missing life cycle information was proposed in [37]. The method reconstructs queue lengths based on a preliminary clustering of the observed times and Bayesian inference of unobserved times using phase-type Markov chains. The difference to our work is that we assume arrival times to be missing, whereas [37] assumes knowledge about arrivals to be given and handles missing information on transit times (between nodes of a queueing network) and activity start times. Hence, the approach in [37] cannot be applied to our scenario of missing arrival times.

Finally, queueing inference adds missing information on queue lengths to an event log and, hence, can be seen as a log repair mechanism. Such mechanisms have been presented for various types of information in logs, see [38], [39]. Notably, the interplay of cases when competing for shared resources was shown to enable the inference of events that are completely missing in an event log [40].

VI. Conclusion

In this paper, we addressed performance analysis for resource-driven processes, for which life cycle information on arrival times is only partially available. To this end, we proposed a methodology for supervised learning that aims at queue length and waiting time prediction. Specifically, we show how features derived by queueing inference under simplistic modelling assumptions are combined with temporal features extracted directly from an event log to achieve a more robust set of features that enables accurate performance prediction. Our empirical analysis reveals that the use of log-based features, in addition to distribution-based features improves the overall performance, especially for the real-world dataset that does not comply with the QIE model assumptions.

In future work, we intend to lift our approach to online scenarios, which requires the integration of online QIE.

References

[1] K. Vergidis, A. Tiwari, and B. Majeed, "Business process analysis and optimization: Beyond reengineering," *IEEE Trans. on Systems, Man, and Cybernetics, Part C*, vol. 38, no. 1, pp. 69–82, 2007.

[2] 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.

[3] W. Van Der Aalst, "Data science in action," in *Process mining*, pp. 3–23, Springer, 2016.

[4] 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.

[5] A. Gal, A. Senderovich, and M. Weidlich, "Challenge paper: data quality issues in queue mining," *Journal of Data and Inf. Qual.*, vol. 9, no. 4, pp. 1–5, 2018.

[6] 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 *BPM*, pp. 306–323, Springer, 2017.

[7] R. Larson, "The queue inference engine: Deducing queue statistics from transactional data," *Manag. Science*, vol. 36, no. 5, pp. 586–601, 1990.

[8] D. J. Bertsimas and L. D. Servi, "Deducing queueing from transactional data: the queue inference engine, revisited," *Operations Research*, vol. 40, no. 3-supplement-2, pp. S217–S228, 1992.

[9] Y. B. Kim and J. Park, "New approaches for inference of unobservable queues," in *Winter Simulation Conference*, pp. 2820–2825, IEEE, 2008.

[10] A. Mandelbaum and S. Zeltyn, "Estimating characteristics of queueing networks using transactional data," *Queueing systems*, vol. 29, no. 1, pp. 75–127, 1998.

[11] G. Bolch, S. Greiner, H. De Meer, and K. S. Trivedi, *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. John Wiley & Sons, 2006.

[12] D. H. Maister *et al.*, *The psychology of waiting lines*. Citeseer, 1984.

[13] C. S. Simpson, M. A. Fisher, M. J. Curtis, A. D. Krahn, H. Abdollah, S. R. Raj, F. J. Brennan, G. J. Klein, and R. Yee, "Correlation of waiting time with adverse events in patients admitted for nonelective permanent pacemaker implantation.," *The Canadian journal of cardiology*, vol. 14, no. 6, pp. 817–821, 1998.

[14] C. Roser, M. Nakano, and M. Tanaka, "Shifting bottleneck detection," in *Winter Simulation Conference*, vol. 2, pp. 1079–1086, IEEE, 2002.

[15] R. A. Nosek Jr and J. P. Wilson, "Queuing theory and customer satisfaction: a review of terminology, trends, and applications to pharmacy practice," *Hospital pharmacy*, vol. 36, no. 3, pp. 275–279, 2001.

[16] N. Munichor and A. Rafaeli, "Numbers or apologies? customer reactions to telephone waiting time fillers.," *Journal of Applied Psychology*, vol. 92, no. 2, p. 511, 2007.

[17] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2009.

[18] J. Abate and W. Whitt, "Transient behavior of the M/M/L queue: Starting at the origin," *Queueing systems*, vol. 2, no. 1, pp. 41–65, 1987.

[19] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[20] I. Verenich, M. Dumas, M. L. Rosa, F. M. Maggi, and I. Teinemaa, "Survey and cross-benchmark comparison of remaining time prediction methods in business process monitoring," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 4, pp. 34:1–34:34, 2019.

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

[22] W. M. Van der Aalst, M. H. Schonenberg, and M. Song, "Time prediction based on process mining," *Inf. Syst.*, vol. 36, no. 2, pp. 450–475, 2011.

[23] A. Rogge-Solti and M. Weske, "Prediction of Remaining Service Execution Time Using Stochastic Petri Nets with Arbitrary Firing Delays," in *ICSOC*, LNCS 8274, pp. 389–403, Springer, 2013.

[24] F. Folino, M. Guarascio, and L. Pontieri, "Mining predictive process models out of low-level multidimensional logs," in *CAiSE*, vol. 8484 of *LNCS*, pp. 533–547, Springer, 2014.

[25] M. Camargo, M. Dumas, and O. González, "Automated discovery of business process simulation models from event logs," *Decis. Support Syst.*, vol. 134, p. 113284, 2020.

[26] 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*, LNCS 5331, pp. 319–336, Springer, 2008.

[27] A. Pika, W. M. P. van der Aalst, C. J. Fidge, A. H. M. ter Hofstede, and M. T. Wynn, "Profiling event logs to configure risk indicators for process delays," in *CAiSE*, LNCS 7908, pp. 465–481, Springer, 2013.

[28] 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," *Inf. Syst.*, vol. 56, pp. 235–257, 2016.

[29] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, *Modelling with generalized stochastic Petri nets*, vol. 292. Wiley, 1995.

[30] G. Balbo, S. C. Bruell, and S. Ghanta, "Combining queueing networks and generalized stochastic petri nets for the solution of complex models of system behavior," *IEEE Trans. Computers*, vol. 37, no. 10, pp. 1251–1268, 1988.

[31] E. Ang, S. Kwasnick, M. Bayati, E. L. Plambeck, and M. Aratow, "Accurate emergency department wait time prediction," *Manufacturing & Service Operations Management*, vol. 18, no. 1, pp. 141–156, 2015.

[32] U. Khurana, H. Samulowitz, and D. S. Turaga, "Feature engineering for predictive modeling using reinforcement learning," in *AAAI*, 2018.

[33] S. Markovitch and D. Rosenstein, "Feature generation using general constructor functions," *Machine Learn.*, vol. 49, no. 1, pp. 59–98, 2002.

[34] Y. Bengio, A. C. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1798–1828, 2013.

[35] A. Senderovich, J. C. Beck, A. Gal, and M. Weidlich, "Congestion graphs for automated time predictions," in *AAAI*, pp. 4854–4861, 2019.

[36] 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*, LNCS 10445, pp. 306–323, Springer, 2017.

[37] A. Senderovich, S. J. J. Leemans, S. Harel, A. Gal, A. Mandelbaum, and W. M. P. van der Aalst, "Discovering queues from event logs with varying levels of information," in *BPM Workshops*, LNBIP 256, pp. 154–166, Springer, 2015.

[38] G. Tello, G. Gianini, R. Mizouni, and E. Damiani, "Machine Learning-Based Framework for Log-Lifting in Business Process Mining Applications," in *BPM*, vol. 11675 of *LNCS*, pp. 232-249, Springer, 2019

[39] A. Rogge-Solti and G. Kasneci, "Temporal anomaly detection in business processes, " in *BPM*, LNCS 8659, pp. 234-249, Springer, 2014.

[40] V. Denisov, D. Fahland, and W.M.P. van der Aalst, "Repairing Event Logs with Missing Events to Support Performance Analysis of Systems with Shared Resources," in *Petri Nets*, pp. 239-259, 2020.

Events Put into Context (EPiC)

Marcus Dees*¶, Bart Hompes†¶, Wil M.P. van der Aalst‡§¶

*Uitvoeringsinstituut Werknemersverzekeringen (UWV), Amsterdam, The Netherlands
† Artifex Consultancy, Eindhoven, The Netherlands
‡ RWTH Aachen University, Aachen, Germany
§ Fraunhofer Institute for Applied Information Technology
¶ Eindhoven University of Technology, Eindhoven, The Netherlands
marcus.dees@uwv.nl, bart.hompes@artifexconsultancy.nl, wvdaalst@pads.rwth-aachen.de

***Abstract*—Business process models can be (re)constructed using event data recorded during the process' execution. Similarly, event data can be used to verify conformance to prescribed behavior and to analyze and improve the underlying processes. However, not all events that are related to a process necessarily relate to its control-flow. Some events occur in the context of the process. In this work, we introduce the concept of *context events* to deal with these types of events. We show how distinguishing between contextual and control-flow events aids process discovery to obtain less complex process models. We demonstrate how visualizing context events on top of process models helps identify points in the process where context events occur often, aiding understanding. We analyze these benefits using two case studies involving real-life processes and event data.**

***Index Terms*—Process Mining, Complex Event Processing, Context events, Business process intelligence**

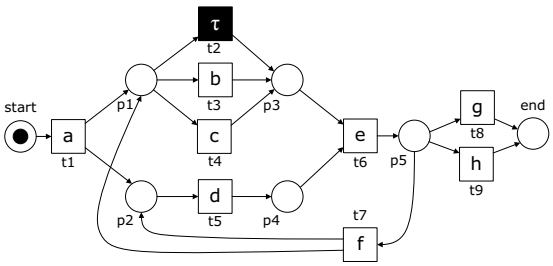
I. Introduction

A process comprises a series of activities performed in order to achieve a specific goal, such as creating a product or delivering a service. A process model describes the *control-flow* of a process, i.e., the order in which its activities should be executed. Organizations use process models in a normative and descriptive manner to define the boundaries of a process and to make sure the process yields the desired result. Modern processes are supported by information systems in which they leave digital footprints in the form of event data. Such events typically represent the occurrence of an instance of an activity. Analyzing event data is valuable because it can reveal previously unknown properties of processes. Process mining is a discipline that analyzes both event data and process models. It comprises three main use cases: process discovery, conformance checking, and process enhancement [1].

In many processes, there exist activities that, while bearing a relation to the process, are not part of its prescribed control-flow. These activities can occur at any time during the process execution. For example, lab tests in a hospital are part of the diagnostic process, but can be executed at any time during the process. Event data representing activities that may occur at any time during the process' execution have proven problematic for most discovery algorithms [2]. Additionally, such behavior complicates conformance checking [3]. In conclusion, it limits process understanding. Existing methods to obtain more precise process models exclude such *contextual behavior* during process discovery. However, by filtering out behavior, the discovered process model will not represent any of the excluded events and lack the complete picture.

It is our aim to both locate *contextual behavior* in a process and to represent it in a complete yet precise process model. To this end, we introduce the concept of *context events*. Context events are events that can be linked to cases in a process, but the activity associated with the event is not part of the prescribed control-flow for the process. As such, context events may *influence* the process, but do not *change* the control-flow state. Context events are a new class of events that are useful to be identified and handled separately from events that are related to the control-flow as defined by a process model.

Fig. 1 introduces a running example of a compensation request process to which we will relate the ideas presented in this paper. The process model for this example is taken from [1]. We use the alphabet a through h to represent the activities of the process. We also introduce three activities, x, y and z, that are related to the process but can not be mapped to an activity in the process' control-flow. These activities represent contact with the customer, which can occur at any given time in the process. There are few alternatives to represent the customer contact activities in our process model. Fig. 2a shows



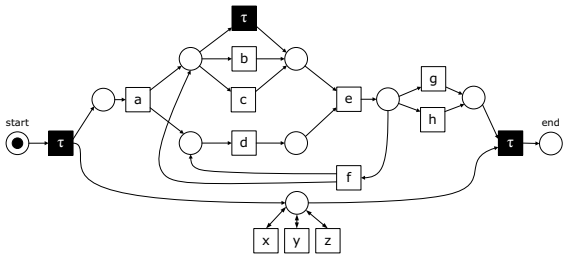
(a) Process model in Petri net notation.

a.	register request	g.	pay compensation
b.	examine thoroughly	h.	reject request
c.	examine casually	h.	reject request
d.	check ticket	x.	incoming call
e.	decide	y.	incoming chat
f.	re-initiate request	z.	incoming complaint

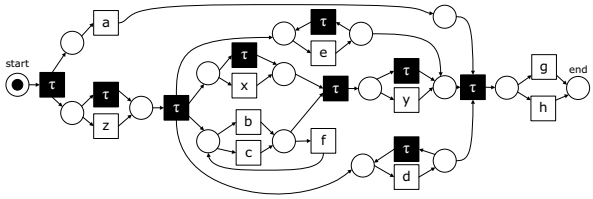
(b) Legend for the transition labels of Fig. 1a

Fig. 1: Running example of a compensation request process.

978-1-7281-9832-3/20/$31.00 ©2020 IEEE
DOI 10.1109/ICPM49681.2020.00020



(a) Fig. 1a with context activities added in a parallel branch.



(b) Discovered process model, using the *Inductive Visual Miner (IVM)* [4], based on a example dataset taken from [1] extended with context events x, y, and z for only 4 out of $1,391$ cases (0.29%).

Fig. 2: Two examples of how the running example can be extended with context activities x, y and z.

how they can be modelled in a parallel branch. This does not allow to pinpoint which activity occurs exactly where in relation to the other activities. Fig. 2b shows a discovered process model based on an example dataset taken from [1]. The dataset is extended with context events x, y, and z for only 4 of the $1,391$ cases (0.29%). The discovered model in Fig. 2b allows activity a to be executed after b,c and d, while in the dataset this never happens. Likewise, the number of occurrences of b plus c is no longer equal to the number of occurrences of d. This shows the influence of only a few events on a process discovery result. Clearly, neither alternative is satisfactory. A novel approach is required that captures the non-control-flow behavior and relates it to the process model.

The remainder of this paper is structured as follows. Section II discusses related work. Section III introduces preliminary definitions. Section IV explains our approach, while in Section V the approach is evaluated using two case studies. Section VI concludes the paper with suggestions for future work.

II. Related Work

Context is used in many different ways. In [1], the context in which events of a process are executed is divided into four categories: case, process, social, and external. These may influence the execution, performance or outcome of a process. In [5], Hompes et al. relate process, case, and activity context information to performance in order to identify possible causal relationships. The multi-perspective process explorer presented in [6] allows the user to visualize context information by selecting elements of a process model and comparing aggregated attribute values between the selected parts of the model. In [3], contextual information consists of control-flow events that occur in the neighborhood of a log pattern.

Our approach to distinguish control-flow events from context events could prove beneficial for other process mining approaches. In both [3] and [7], context is used during process discovery. However, no distinction is made between control-flow events and events that occur in the context of the process. Another way of improving the result of process discovery, is by filtering out undesired behavior from an event log before applying a discovery algorithm. An approach for this using information theory concepts is presented in [8], while in [9] the probability of an activity occurring in the context of other activities is used. Finally, a different approach to improving the result of discovery is the repair approach presented in [10]. A given process model is repaired to reflect the behavior present in an event log from the same process. Both model discovery and model repair techniques can benefit from our approach, as it helps reduce the number of activities and relations between those activities, hence reducing the complexity of the respective task, as shown in Fig. 2.

Visualization, used by many process mining techniques, can be improved by separating context information and control-flow information. As mentioned above, the multi-perspective process explorer [6] allows to aggregate the values of case and event attributes in the context of a process model. However, it does not allow to show contextual events in relation to a process model. Similarly, in [11], an artifact-centered approach is used to show multiple related (sub-)processes and how they interact with each other. Since context events are not necessarily related to one another, applying this approach in a process with contextual behavior does not lead to meaningful results. Work by de Leoni et al. creates movies in which the state of each process activity is calculated at each moment in time [12]. The result is plotted on top of a process model. However, there is no option to show events that are not part of the control-flow, i.e., that are not part of the process model in the movie. Only the number of unmapped events at any time during the process is shown in the process movie.

In [13], four questions are raised regarding the overlap between the fields of complex event processing (CEP) and process mining. Our work can be applied towards the challenge of process mining sensor events. Identifying control-flow-related sensor events, while keeping track of when other (contextual) sensor events take place, helps identify those sensor events that can be transformed into activities. Mandal et al. use an example from the logistics domain [14]. Goods are transported from a warehouse to a customer. While on route, incidents may happen that require recalculating the route. Knowing when incidents (context events) occur in relation to the transportation process (control-flow) facilitates the adaptation of the process, reducing the influence of incidents on the delivery time.

III. Preliminaries

The research reported in this paper builds on a body of existing research in process mining. Although our approach is generic and applicable to any modelling language (e.g., BPMN, EPC, etc.), we opt for Petri nets due to their simple and clear semantics. Our approach uses alignments of Petri nets and event logs for the localisation of context events.

Processes supported by information systems leave a digital footprint in event logs. Events have properties identifying different aspects of the event, such as a timestamp or the corresponding activity. Events from the same process instance are grouped into cases, and multiple cases form an event log.

Definition 1 (Event, Event attributes). *Let $\mathcal{E}$ be the universe of events, i.e., the universe of unique event identifiers. Let $\mathcal{Q}$ be the universe of event properties and let $\mathcal{V}$ be universe of event property values. Let $\pi \in \mathcal{Q} \rightarrow (\mathcal{E} \nrightarrow \mathcal{V})$ be the event property function. For any property $q \in \mathcal{Q}, \pi(q)$ $(denoted\ \pi_q)$ is a partial function mapping events onto values. If $\pi_q(e) = v$, then event $e \in \mathcal{E}$ has a property $q \in \mathcal{Q}$ and the value of this property is $v \in \mathcal{V}$. If $e \notin dom(\pi_q)$, then event e does not have property q and we write $\pi_q(e) = \perp$. Let $\mathcal{A}$ be the universe of activity labels, $label \in \mathcal{Q}$, $dom(\pi_{label}) = \mathcal{E}$, and $rng(\pi_{label}) = \mathcal{A}$, i.e., every event has a label property value.*

Definition 2 (Case, Trace, Event log). *A* case $c \in \mathcal{E}^*$ *is a sequence of events, i.e., $c = \langle e_1, e_2, \ldots, e_n \rangle$. A* trace $\sigma \in \mathcal{A}^*$ *is a sequence of activity labels. We define function $trace$ to project a case onto its trace, i.e., $trace : \mathcal{E}^* \rightarrow \mathcal{A}^*$ and $trace(c) = \langle \pi_{label}(e_1), \pi_{label}(e_s), \ldots, \pi_{label}(e_n) \rangle$. Let $\mathcal{L}$ be the universe of event logs. An* event log $L \in \mathcal{L}$ *is a set of cases, i.e., $L \subseteq \mathcal{E}^*$.*

Definition 3 (Petri net, Petri net edge). *Let $\mathcal{N}$ be the universe of Petri nets. Let $\mathcal{P}$ be the universe of places. Let $\mathcal{T}$ be the universe of transitions. Let $P \subseteq \mathcal{P}$ be a set of places, $T \subseteq \mathcal{T}$ be a set of transitions, and $F \subseteq (P \times T) \cup (T \times P)$ be a flow relation between places and transitions (and between transitions and places). A Petri net $N \in \mathcal{N}$ is a tuple $N = (P, T, F)$. A Petri net edge $f \in F$ is an input edge if $f \in P \times T$ and an output edge if $f \in T \times P$. We define function $label$ as a transition labeling function, i.e., $label \in \mathcal{T} \rightarrow \mathcal{A} \cup \{\tau\}$.*

Definition 4 (Marking, System net). *A marking M is a multiset of places, i.e., $M \in \mathbb{B}(\mathcal{P})$. Let $\mathcal{S}$ be the universe of system nets. A system net $S \in \mathcal{S}$ is a triplet (N, M_{init}, M_{final}) where $N \in \mathcal{N}$ is a Petri net, $M_{init} \in \mathbb{B}(P)$ is the initial marking, and $M_{final} \in \mathbb{B}(P)$ is the final marking.*

Figure 1a shows an example Petri net. Visually, a Petri net consists of squares, circles, and directed edges, which represent the transitions, places, and flow relations, respectively. Transitions represent process activities. The only exceptions are the *invisible* transitions labeled τ, which do not represent pieces of the process' work but are necessary to properly model some types of routing of the process. Places may contain tokens; whereas the structure of the Petri net never changes, tokens are created and consumed. A transition is enabled (the activity it represents is allowed to occur at the current state) if and only if at least one token exists in each input place of the transition. By firing (i.e., executing) a transition, a token is consumed from each input place and a token is produced for each output place. The state of a Petri net is uniquely determined by the distribution of tokens over places, which is denoted as its *marking*. For the Petri net in Fig. 1a, the initial marking is place *start*, and the final marking is place *end*. A complete firing sequence is a sequence of transitions leading from the initial marking to the final marking, indicating a complete execution of a process instance. The set of all complete firing sequences of a system net S is denoted by Ψ_S. For further information on Petri nets in relation to process mining, readers are referred to [1].

Conformance checking aims to verify whether observed behavior recorded in an event log matches behavior described by a process model. The notion of *alignments* provides a robust approach to conformance checking, which makes it possible to pinpoint the deviations causing nonconformity [15]. Building such alignments between an event log and a process model is not trivial, since the log may deviate from the model at an arbitrary number of places. We need to relate "moves" in the log to "moves" in the model. However, it may be that some of the moves in the log cannot be mimicked by the model and vice versa. In other words, not all traces in an event log may be reproducible by the corresponding Petri net, i.e., not all traces may correspond to a complete firing sequence.

Definition 5 (Alignment move). *Let $\mathcal{M}$ be the universe of alignment moves, i.e., the universe of unique alignment move identifiers. We define function λ as follows. Given $m \in \mathcal{M}$:*

- $\lambda_{event}(m) \in \mathcal{E} \cup \{\gg\}$,
- $\lambda_{label}(m) \in \mathcal{A} \cup \{\tau\}$,
- $\lambda_{trans}(m) \in \mathcal{T} \cup \{\gg\}$,

where $\gg$ identifies the absence of respectively an event or a transition, such that

- $\lambda_{event}(m) = \gg \implies \lambda_{trans}(m) \in \mathcal{T}$,
- $\lambda_{trans}(m) = \gg \implies \lambda_{event}(m) \in \mathcal{E}$,
- $\lambda_{event}(m) \neq \gg \implies \lambda_{label}(m) = \pi_{label}(\lambda_{event}(m))$,
- $\lambda_{trans}(m) \neq \gg \implies \lambda_{label}(m) = label(\lambda_{trans}(m))$.

Definition 6 (Alignment, Alignment set). *An alignment $\gamma \in \mathcal{M}^*$ is a finite sequence of alignment moves such that each alignment move appears only once, i.e., $1 \leq i < j \leq |\gamma| : \gamma(i) \neq \gamma(j)$. An alignment set is $\Gamma \subseteq \mathcal{M}^*$ such that each alignment move appears only once. $\mathcal{U}_\gamma$ is the universe of alignments and $\mathcal{U}_\Gamma$ the universe of alignment sets.*

For a given alignment move $m \in \mathcal{M}$, m is considered a synchronous move when $\lambda_{event}(m) \neq \gg \wedge \lambda_{trans}(m) \neq \gg$, a model move when $\lambda_{event}(m) = \gg$ and $\lambda_{label}(m) \neq \tau$, a log move when $\lambda_{trans}(m) = \gg$ and an invisible move when $\lambda_{label}(m) = \tau$. For every alignment $\gamma \in \mathcal{U}_\gamma$ of a system net $S \in \mathcal{S}$ and a case $c \in \mathcal{E}^*$, the projection on λ_{event} yields c, and the projection on λ_{trans} yields a path in Ψ_S.

Multiple alignments are possible for the same case and model. The aim is to find a complete alignment with a minimal number of deviations, also known as an *optimal alignment* [15]. For the sake of space, we assume here that all deviations (i.e., model moves for visible transitions and log moves) have unit cost. In [15], Aalst et al. show how this assumption can be removed. Different alignment techniques exist, which we generalize by the alignment function.

Definition 7 (Alignment function). *The alignment function $align \in \mathcal{L} \times \mathcal{S} \rightarrow \mathcal{U}_\Gamma$ aligns an event log with a system net and returns an alignment set consisting of one optimal alignment for each case in the event log.*

Clearly, different cases may have different (optimal) alignments as they contain different events and deviations. For example, Fig. 3 shows three possible alignments for trace $\langle a, e, b, d, g\rangle$ and the Petri net (system net) in Fig. 1a. Here, γ_1 contains two deviations: one log move (m_{12}) and one model move (m_{15}). Furthermore, γ_2 contains one invisible move (m_{22}), a model move (m_{23}), and two log moves (m_{25} and m_{26}). Lastly, γ_3 contains an invisible move (m_{32}), two model moves (m_{33} and m_{37}), and one log move (m_{34}). Since no alignment exists with less than two asynchronous moves for visible transitions, γ_1 is an optimal alignment for the trace.

IV. Putting Events into Context

In this paper, we argue that not all events should be treated equally. We distinguish two main types of events: control-flow events and context events. We aim to visualize the process model using both event types. The concept of alignments is used to identify where in a process model context events occur. This requires context events to be identified (Section IV-A) and that a mapping is created between the context events and the process model (Section IV-B).

A. Identifying Context Events

In order to identify different types of events, we introduce an event property $type \in \mathcal{Q}$ with $dom(\pi_{type}) = \mathcal{E}$. In other words, every event has an event type. The default value is *control-flow*. This implies that an event is part of the control-flow of the process. During analysis, an analyst may define other *event types* to relate to context events. For example, a purchasing process typically consists of activities such as placing an order, making a payment, receiving a request, and sending goods. These events would be of type *control-flow*. Other events may occur that may *influence* the process, but do not *change* the state of the process. For example, a customer may call to inquire the status of the order, may file a complaint, or may visit the website to check the current prices. These events can be defined as events of type *context*. Furthermore, multiple event types may be identified to distinguish different types of contextual events (e.g., *call*, *complaint*, *website visit*).

$$\gamma_1 = \left|\begin{array}{c|c|c|c|c|c} m_{11} & m_{12} & m_{13} & m_{14} & m_{15} & m_{16} \\ e_{11} & e_{12} & e_{13} & e_{14} & \gg & e_{15} \\ a & e & b & d & e & g \\ t_1 & \gg & t_3 & t_5 & t_6 & t_8 \end{array}\right|$$

$$\gamma_2 = \left|\begin{array}{c|c|c|c|c|c|c} m_{21} & m_{22} & m_{23} & m_{24} & m_{25} & m_{26} & m_{27} \\ e_{11} & \gg & \gg & e_{12} & e_{13} & e_{14} & e_{15} \\ a & \tau & d & e & b & d & g \\ t_1 & t_2 & t_5 & t_6 & \gg & \gg & t_8 \end{array}\right|$$

$$\gamma_3 = \left|\begin{array}{c|c|c|c|c|c|c|c} m_{31} & m_{32} & m_{33} & m_{34} & m_{35} & m_{36} & m_{37} & m_{38} \\ e_{11} & \gg & \gg & e_{12} & e_{13} & e_{14} & \gg & e_{15} \\ a & \tau & d & e & b & d & e & g \\ t_1 & t_2 & t_5 & \gg & t_3 & t_5 & t_6 & t_8 \end{array}\right|$$

Fig. 3: Three example alignments of trace $\langle a, e, b, d, g\rangle$ with the system net in Fig. 1a. Columns represent alignment moves. The rows represents the alignment move identifier, the event identifier, the label, and the transition, from top to bottom.

Process models describe the lifecycle of a single process instance, i.e., a case. As such, events that are not linked to a case cannot be logically represented on a process model. To analyze context events, they need to be included in an event log and linked to a case. A connection is to be made between the context events and the control-flow events. Such connections can be made based on shared event properties (e.g., order id) or on the time period in which the events occurred.

Figure 4 shows the alignments for the traces $\langle a, d, c, g, h\rangle$ and $\langle a, d, h\rangle$ with the Petri net in Fig. 1a. Context events labeled x, y, and z are inserted. Since there are no transitions in the Petri net labeled x, y, or z, these context events thus become log moves in the alignment. In order to distinguish log moves for context events (i.e., events of type *context*) from regular log moves (i.e., log moves for events of type *control-flow*), we introduce alignment move types. The type of the move equals the type of the event. If the move does not correspond to an event, the move is of type *control-flow*.

Definition 8 (Alignment move type). *Let $m \in \mathcal{M}$ be an alignment move. We extend function λ such that:*

- $\lambda_{event}(m) \neq \gg \implies \lambda_{type}(m) = \pi_{type}(\lambda_{event}(m))$
- $\lambda_{event}(m) = \gg \implies \lambda_{type}(m) =$ *control-flow*

$$\gamma_4 = \left|\begin{array}{c|c|c|c|c|c|c|c} m_{41} & m_{42} & m_{43} & m_{44} & m_{45} & m_{46} & m_{47} & m_{48} \\ e_{41} & e_{42} & e_{43} & e_{44} & e_{45} & \gg & e_{46} & e_{47} \\ a & d & x & c & y & e & g & h \\ t_1 & t_5 & \gg & t_4 & \gg & t_6 & t_8 & \gg \end{array}\right|$$

$$\gamma_5 = \left|\begin{array}{c|c|c|c|c|c|c|c} m_{51} & m_{52} & m_{53} & m_{54} & m_{55} & m_{56} & m_{57} & m_{58} \\ e_{51} & e_{52} & e_{53} & e_{54} & \gg & \gg & e_{55} & e_{56} \\ a & d & x & y & \tau & e & h & z \\ t_1 & t_5 & \gg & \gg & t_2 & t_6 & t_9 & \gg \end{array}\right|$$

Fig. 4: Example of alignments with context moves highlighted in gray. In γ_4 (trace $\langle a, d, x, c, y, g, h\rangle$), m_{43}, and m_{45} are of type context. In γ_5 (trace $\langle a, d, x, y, h, z\rangle$), m_{53}, m_{54}, and m_{58} are of type context.

B. Visualizing Context Moves

In existing literature, when projecting alignment data onto Petri nets, model moves are commonly associated with transitions and log moves with places [1]. As such, we have chosen to *associate context moves* (i.e., moves of type context) *to Petri net edges* as to avoid ambiguity about information mapped onto the process model. This design decision has several consequences that need to be addressed.

Firstly, to determine on which edges a context move should be plotted, we need to determine through which transitions a case travels, i.e., what its corresponding firing sequence is. Since context moves occur in-between control-flow moves, it is enough to determine the preceding and succeeding control-flow move in the alignment, for each context move. To do this, we define function *enrich*. Each context move is mapped to exactly one control-flow predecessor move and

one control-flow successor move. If the context move is the first (respectively last) move in the alignment, the predecessor (respectively successor) does not exist and we write $\gg$. Figure 5 shows the results of applying function *enrich* to γ_4 and γ_5.

Definition 9 (Enrich function). *We define function* $enrich$ *to enrich each context move in an alignment with it's preceding and succeeding control-flow alignment move, i.e.:* $enrich : \mathcal{U}_\gamma \rightarrow \mathcal{M} \times (\mathcal{M} \cup \{\gg\}) \times (\mathcal{M} \cup \{\gg\})$.

$$\begin{aligned} enrich(\gamma_4) = &\quad \{(m_{43}, m_{42}, m_{44}), (m_{45}, m_{44}, m_{46})\} \\ enrich(\gamma_5) = &\quad \{(m_{53}, m_{52}, m_{55}), (m_{54}, m_{52}, m_{55}), \\ &\quad (m_{58}, m_{57}, \gg)\} \end{aligned}$$

Fig. 5: Results of $enrich(\gamma_4)$ and $enrich(\gamma_5)$.

Secondly, since transitions might have multiple input and output edges, if we would associate a context move with every outgoing edge of the transition corresponding to the preceding control-flow move and every incoming edge of the transition corresponding to the succeeding control-flow move, we risk increasing the total number of context moves associated with the process model. To avoid this, we assign a weight to every context move. If a context move is mapped onto n edges, this weight is calculated as $\frac{1}{n}$.

Thirdly, the control-flow moves that precede or succeed a context move may be model moves, i.e., the labels represented by their corresponding transitions were expected in the model, but they did not occur in the event log. In this case, we cannot be entirely certain that the context move actually occurred between these two model transitions. We argue that, depending on whether only one of the surrounding control-flow moves in the alignment is a model move, or both, the certainty decreases. This is captured by the function $cert$. Given a context move, its predecessor, and its successor, the certainty is calculated as shown in Fig. 6.

$$cert(m_{cont}, m_{pred}, m_{succ}) = \begin{cases} 0.25 & \lambda_{trans}(m_{pred}) = \gg \wedge\ \lambda_{trans}(m_{succ}) = \gg \\ 0.50 & \lambda_{trans}(m_{pred}) = \gg \wedge\ \lambda_{trans}(m_{succ}) \neq \gg \\ 0.50 & \lambda_{trans}(m_{pred}) \neq \gg \wedge\ \lambda_{trans}(m_{succ}) = \gg \\ 1.00 & \lambda_{trans}(m_{pred}) \neq \gg \wedge\ \lambda_{trans}(m_{succ}) \neq \gg \end{cases}$$

Fig. 6: Calculation of the certainty that a context move has occurred in between two control-flow moves.

We choose this way of calculating weight and certainty for simplicity and ease of use. Other aspects of the model, the event log, or their alignment can be taken into account in order to calculate how certain one can be of a context move occurring between any two transitions of the model. For example, the prefix or postfix of the trace, or even the number of optimal alignments could be used.

Once the context moves are identified and enriched with their surrounding control-flow moves, they can be mapped onto a Petri net model. Algorithm 1 shows how context moves are mapped to Petri net edges. Context moves are plotted as *close as possible* to the location in the process model where they occurred, i.e., as close as possible to the transitions corresponding to the control-flow moves immediately before and after the context move. When a context move is positioned in between two transitions that have a route between them, then it is mapped on the two edges connecting the transitions in the Petri net. In Algorithm 1 this is expressed in the situation where *overlap* exists between the Petri net places related to both transitions. When no such overlap exists, the context move is plotted on all outgoing edges of the transition corresponding to the preceding control-flow move and on all incoming edges of the transition corresponding to the succeeding control-flow move. When the context move does not have a preceding control-flow move, i.e., it occurred before the first control-flow move, then the context move is plotted on the incoming edges of the transition corresponding to the succeeding control-flow move. Finally, when the context move does not have a succeeding control-flow move, it is plotted on the outgoing edges of the transition corresponding to the preceding control-flow move.

Algorithm 1: Map Context Moves onto Petri net edges

Input: $N = (P, T, F) \in \mathcal{N}$
Input: $\Gamma \in \mathcal{U}_\Gamma$
Output: $EF \subseteq F \times \mathcal{M} \times \mathbb{R} \times \mathbb{R}$

$EF \leftarrow \emptyset$
foreach $\gamma \in \Gamma$ **do**
  $EM \leftarrow enrich(\gamma)$
  foreach $(m_{cont}, m_{pred}, m_{succ}) \in EM$ **do**
    $ct \leftarrow cert(m_{cont}, m_{pred}, m_{succ})$
    $t_p \leftarrow \lambda_{trans}(m_{pred})$
    $t_s \leftarrow \lambda_{trans}(m_{succ})$
    $f_{out} \leftarrow \{(t_p, p_p) \in F\}$
    $f_{in} \leftarrow \{(p_s, t_s) \in F\}$
    $overlap \leftarrow \{(t_p, p_p) \in f_{out} \mid \exists (p_s, t_s) \in f_{in} \wedge p_s = p_p\} \cup \{(p_s, t_s) \in f_{in} \mid \exists (t_p, p_p) \in f_{out} \wedge p_p = p_s\}$
    if $|f_{out}| = 0 \wedge |f_{in}| > 0$ **then**
      foreach $f \in f_{in}$ **do**
        $EF \leftarrow EF \cup \{(f, m_{cont}, \frac{1}{|f_{in}|}, ct)\}$
    else if $|f_{out}| > 0 \wedge |f_{in}| = 0$ **then**
      foreach $f \in f_{out}$ **do**
        $EF \leftarrow EF \cup \{(f, m_{cont}, \frac{1}{|f_{out}|}, ct)\}$
    else if $|overlap| > 0$ **then**
      foreach $f \in overlap$ **do**
        $EF \leftarrow EF \cup \{(f, m_{cont}, \frac{1}{|overlap|}, ct)\}$
    else if $|overlap| = 0$ **then**
      foreach $f \in f_{in} \cup f_{out}$ **do**
        $EF \leftarrow EF \cup \{(f, m_{cont}, \frac{1}{|f_{in}|+|f_{out}|}, ct)\}$
return EF

Consider context move m_{43} of alignment γ_4 in Fig. 4. This move corresponds to event e_{43} (label x), and is positioned in between m_{42} (label d, transition t_5), and m_{44} (label c, transition t_4). Both m_{42} and m_{44} are synchronous alignment moves. This implies that the certainty that e_{43} occurred between t_5 and t_4 is equal to 1. In the Petri net (shown in Fig. 1a) there are no shared places between the outgoing edges from transition t_5 and the incoming edges from transition t_4. Following Algorithm 1, context event e_{43} is thus related to edge (p_2, t_5) and edge (p_1, t_4), both with a weight of $\frac{1}{2}$.

V. Evaluation

This section reports on an evaluation of our technique based on two case studies. The first case study, presented in Section V-A, is based on a hospital process in which patients are tested for sepsis. These tests can be executed at any time during the patients stay in the hospital. The second case study, in Section V-B, shows an insurance claim process in which customers can call the insurance company to ask questions regarding their claim. The approach has been implemented in ProM. [1]

A. Case Study Emergency Room Sepsis Process Discovery

This section illustrates how our technique can be beneficial for process discovery [1]. When selected events in the event log are marked as context events, they can be kept aside while executing the process discovery process. Using less activities in process discovery will lead to a less complex result. After a model is discovered, the context events can be plotted on top of this model, highlighting the locations where the context events are most likely to occur. Visualizing context events in this way can identify the best locations for modifying the process model by adding *context activities*, which relates to model repair.

The process used in the case study in this section was introduced by Mannhardt et al. in [16]. The data is publicly available through [17]. The data describes cases that follow a process starting in the emergency room of a hospital and ending when the patient is released from the hospital. The data focuses on activities to detect and handle sepsis. Sepsis is the presence in tissues of harmful bacteria and their toxins, typically through infection of a wound. The data consists of 16 activities leading to 15,214 events for 1,050 cases. The event log has 846 different execution paths (trace variants). Three activities are related to lab tests, i.e., *CRP*, *Lactic Acid* and *Leucocytes*. These activities can be performed anywhere during the process. When events for these activities are removed from the event log, we still have the same 1,050 cases, however, the number of variants is reduced to 182. A reduction of 664 variants by only removing these three activities highlights the complexity caused by considering those events as control-flow events. The same reduction in complexity can be observed when Fig. 7 is compared to Fig. 8. Both figures show the result of a process discovery analysis using the *Directly Follows Miner (DFM)* [18]. While the number of activities is reduced by only three, the number of connections between the transitions of the process model is reduced ten-fold. On the other hand, we have removed information about the execution of the lab tests and it is no longer visible when they can occur.

To provide insights into when the lab tests are executed, we use the hand-made Petri net presented in [16]. Next, we mark the three lab test events in the event log as context events. The context enriched event log is aligned with the model and finally, the lab test events are plotted on top of the model. The resulting enriched process model is shown in Fig. 9. The figure shows the sum of the weights of the

[1]ProM is an extensible tool that supports a wide variety of process mining techniques through plug-ins, see http://www.promtools.org. Our technique is part of the *ContextEvents* package.

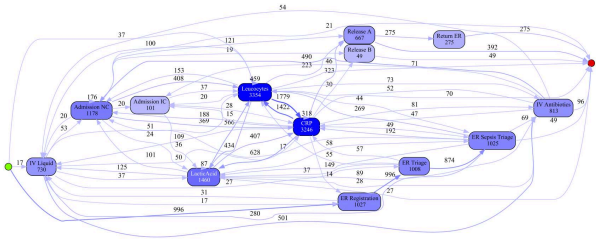


Fig. 7: Result of the DFM miner [18] on sepsis event log [17], using all events and the path slider is set to 80%.

Fig. 8: Result of the DFM miner [18] on sepsis event log [17], not using the lab events and the path slider is set to 80%.

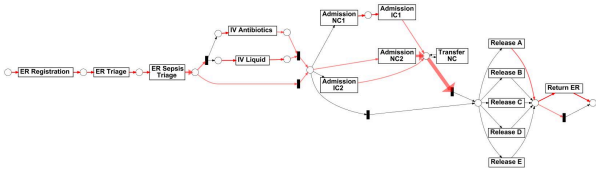


Fig. 9: Absolute context heatmap of sepsis process lab tests.

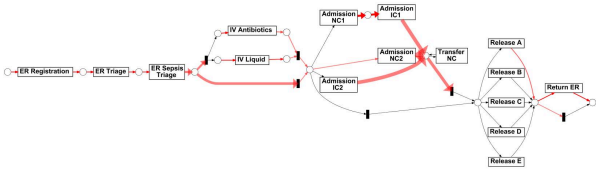


Fig. 10: Relative context heatmap of sepsis process lab tests.

context events on each edge, highlighted in a red color scale. An edge is colored black when the edge does not have any context events mapped to it. The width of the edge indicates the *sum of the weights of all context events* that are related to the edge. The red color intensity of an edge indicates the sum of the certainties of the context events related to that edge. We refer to this visualisation as an *absolute context heat map*. We use the term heatmap because the figure shows the magnitude of a phenomenon, i.e., the occurrence of context events. The variation in color gives a visual cue to the reader about how certain we are about the magnitude.

Fig. 9 shows that most lab tests are executed after *Admission NC2*. Since *Admission NC* is the activity with the highest frequency in the event log it is as expected that lab tests have a high occurrence rate at this point in the process model. We create a so-called *relative context heatmap* to put the cases with context events related to an edge in perspective of the total number of cases that traverse that edge, i.e., the percentage of cases with a context event related to that edge is calculated and plotted.

If we look at the *relative context heatmap*, presented in Fig. 10, a different insight emerges. Relatively, most lab tests

are executed after an admission to a normal or intensive care department, i.e., *Admission NC* and *Admission IC*. Lab tests also occur immediately following the *ER Sepsis Triage* activity when both *IV Antibiotics* and *IV Liquid* are not executed.

We can take this analysis a step further by filtering on cases for which a selected event occurs. In Fig. 11 the data set is filtered on cases having an event that is aligned to the *Admission IC1* transition, and in Fig. 12 only cases that have an event that is aligned to the *Admission IC2* transition. Fig. 11 shows that from *Admission NC1* to *Admission IC1* and immediately after *Admission IC1*, relatively often a lab test is executed. Respectively for 54% of cases after *Admission NC1* and 100% after *Admission IC1*. Fig. 12 shows that after *Admission IC2*, *always* a lab test is executed. Combining Fig. 11 and Fig. 12 reveals that after the admission to an intensive care department a lab test is always executed. Adding a mandatory labtest activity to the process model, immediately after both the *Admission IC* activities, would make it possible to check if the labtest is always executed or not.

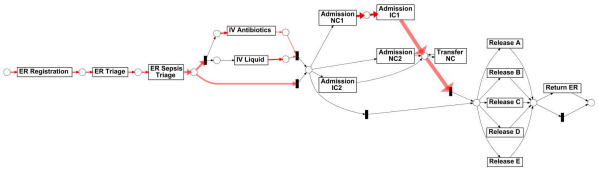


Fig. 11: Relative context heatmap of sepsis process lab tests, filtered on cases that have activity *AdmissionIC1*.

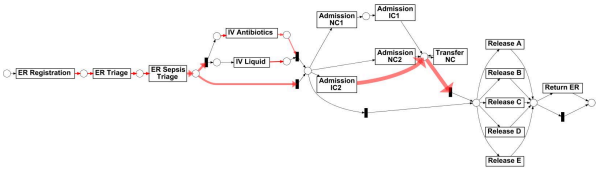


Fig. 12: Relative context heatmap of sepsis process lab tests, filtered on cases that have activity *AdmissionIC2*.

B. Case Study UWV Claim Process

UWV is the social security institute of the Netherlands and responsible for the implementation of a number of employee related insurances. The case study focuses on the unemployment benefits claim process of UWV. When employees become unemployed, they may be entitled to the benefits. Employees have to file a claim at UWV, which then decides whether they are entitled to benefits. When claims are accepted, employees receive benefits with a regular frequency until they find a new job or the maximum period for their entitlements is reached. UWV refers to employees who are making use of their services as customers, therefore we use the term customer in the remainder of the paper.

The unemployment benefits claim process starts at UWV when a claim is received. First, the customer is sent a change form, because the customer is obliged to notify UWV of any changes during the claim handling process. Next a check is performed whether all required information is available. If this is not the case, the customer can be requested to provide the missing information. The request can be done by two types of letters or a phone call. When the information is received, UWV registers a *Document IN* event. If needed a reminder can be send to the customer. When all information is received the decision is made. Finally, the customer is notified of the decision by the appropriate letter.

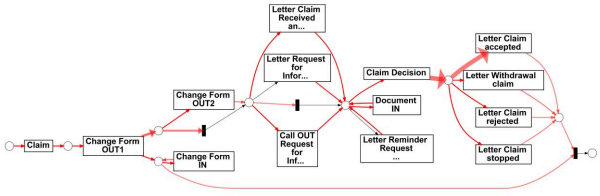


Fig. 13: Absolute context heatmap for UWV claim process.

Before, during and after the claim handling process the customer can contact the UWV call center. Every contact is registered as a *Call IN* event including a description of the questions asked by the customer. The process model used in this case study is obtained from a process specialist at UWV. Fig. 13 shows the absolute context heatmap for the UWV claim process. Incoming calls occur during the whole process execution as can be seen from almost all edges being colored red and not being thin. The most incoming calls occur after the decision is made and before the acceptance letter is sent. Most customers get accepted (on average 80%). The second most frequent location of calls is immediately after the first *Change Form OUT*, which occurs at the start of the process.

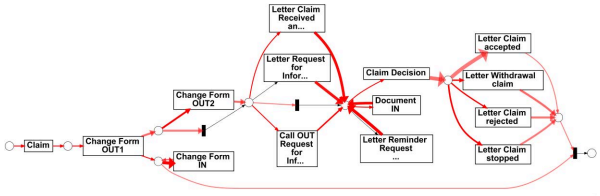


Fig. 14: Relative context heatmap for UWV claim process.

The relative context heatmap in Fig. 14 shows there are several locations that trigger similar amounts of incoming calls. The first location is again between the decision and the acceptance letter. Also, questions come in after information was requested from the customer by sending a letter. Interestingly, when information was requested by phone, a relatively smaller number of customers ask questions. This gives rise to the insight that contacting customers by phone could be beneficial for UWV to reduce the number of incoming calls. Next, just before receiving a *Change Form IN*, questions come in. Finally, after receiving a negative decision, withdrawing the claim or when the claim is stopped, more questions are triggered compared to when the claim is accepted. Providing a customer with adequate information regarding the decision could help reduce the number of customers calling UWV. This analysis shows that knowing when customers call is helpful when advising UWV on how the process could be improved from both the customer and UWV's perspective.

VI. Conclusion

The technique introduced in this paper uses existing process mining concepts such as event logs, process models, and alignments to introduce the concept of context events: events that are part of the event log and can be linked to cases in a process, but are not part of the process' control-flow. Highlighting such events on a process model separately from the control-flow activities provides added value by showing novel insights and facilitating process discovery.

The emergency room sepsis case study in Section V-A shows that keeping selected events aside while doing process discovery leads to less complex results. The events that are first kept aside, can, with our technique, be once more related to the process model. In this way the information contained in the events is used to improve the discovered model. In the UWV case study in Section V-B, customers can call UWV while their unemployment benefits claim is being processed. Identifying the locations in the process model when relatively the most customers call, gives input for improving the customer journey.

We foresee several directions for further research on this topic. For example, up until now, we have only looked at atomic events, i.e., events without duration. Instead, we could also look at durative events, i.e., events that have a start and a complete timestamp. These types of events identify a period in which they are active. Examples of these types of durative event are the caseload of a system and the happiness level of a customer during the process execution. One option of plotting these periods on top of a process model is to use a heatmap, reflecting the average caseload of the system or the average happiness of the customers when the process was executed. In this way, a more complex context can be related to the model.

In this paper, we have shown how to plot a single context event type in relation to a process model. Research could be directed to finding ways to plot multiple context classes on the same process model. This should be done in a way that is still comprehensible to the users of the visualization. A simple yet effective direction would be to use icons attached to the Petri net edges. Different icons may represent different event types.

Instead of having an analyst identify context events, the labeling could be done in a semi- or fully automatic way. One approach would be to use the techniques developed by Tax et al. [8] to identify so-called chaotic activities.

Another possible direction for future work is to use data attributes when plotting context events. In this way, context events can be conditionally plotted or guards could be derived indicating under which circumstances context events occur. Inversely, context events themselves can become input for explanatory data analysis. For example in the work work by de Leoni et al. [19] on process predictions. In the work by Van der Aalst et al. [20] on visualizing token flows using interactive performance spectra the context events can be used as a new classifier to explain the process behavior.

Finally, while we use calculation heavy alignments in our current approach, it is also possible to use the more performance friendly token-based replay.

Acknowledgment

Creating this paper would not have been possible without the help of Maikel Leemans.

References

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

[2] M. F. Sani, S. J. van Zelst, and W. M. P. van der Aalst, "Repairing outlier behaviour in event logs," in *Business Information Systems - 21st International Conference, BIS, Berlin, Germany, Proceedings*, 2018.

[3] X. Lu, D. Fahland, R. Andrews, S. Suriadi, M. T. Wynn, A. H. M. ter Hofstede, and W. M. P. van der Aalst, "Semi-supervised log pattern detection and exploration using event concurrence and contextual information," in *OTM 2017 Proceedings, Part I*, 2017, pp. 154–174.

[4] 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 *Business Process Management Workshops - BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers*, 2013, pp. 66–78.

[5] B. F. A. Hompes, J. C. A. M. Buijs, and W. M. P. van der Aalst, "A generic framework for context-aware process performance analysis," in *On the Move to Meaningful Internet Systems: OTM 2016 Conferences Proceedings*, 2016, pp. 300–317.

[6] F. Mannhardt, M. de Leoni, and H. A. Reijers, "The multi-perspective process explorer," in *Proceedings of the BPM Demo Session*, 2015.

[7] P. M. Dixit, H. M. W. Verbeek, J. C. A. M. Buijs, and W. M. P. van der Aalst, "Interactive data-driven process model construction," in *Conceptual Modeling - 37th International Conference, ER 2018, Xi'an, China, October 22-25, 2018, Proceedings*, 2018, pp. 251–265.

[8] N. Tax, N. Sidorova, and W. M. P. van der Aalst, "Discovering more precise process models from event logs by filtering out chaotic activities," *J. Intell. Inf. Syst.*, vol. 52, no. 1, pp. 107–139, 2019. [Online]. Available: https://doi.org/10.1007/s10844-018-0507-6

[9] M. F. Sani, S. J. van Zelst, and W. M. P. van der Aalst, "Repairing outlier behaviour in event logs using contextual behaviour," *Enterp. Model. Inf. Syst. Archit. Int. J. Concept. Model.*, vol. 14, pp. 5:1–5:24, 2018. [Online]. Available: https://doi.org/10.18417/emisa.14.5

[10] D. Fahland and W. M. P. van der Aalst, "Model repair - aligning process models to reality," *Inf. Syst.*, vol. 47, pp. 220–243, 2015. [Online]. Available: https://doi.org/10.1016/j.is.2013.12.007

[11] M. L. van Eck, N. Sidorova, and W. M. P. van der Aalst, "Discovering and exploring state-based models for multi-perspective processes," in *Business Process Management Conference Proceedings*, 2016.

[12] M. de Leoni, S. Suriadi, A. H. M. ter Hofstede, and W. M. P. van der Aalst, "Turning event logs into process movies: animating what has really happened," *Software and System Modeling*, vol. 15, no. 3, pp. 707–732, 2016.

[13] P. Soffer, A. Hinze, A. Koschmider, H. Ziekow, C. D. Ciccio, B. Koldehofe, O. Kopp, A. Jacobsen, J. Sürmeli, and W. Song, "From event streams to process models and back: Challenges and opportunities," *Information Systems*, 2017.

[14] S. Mandal, M. Hewelt, and M. Weske, "A framework for integrating real-world events and business processes in an iot environment," in *OTM 2017 Conferences Proceedings, Part I*, 2017, pp. 194–212.

[15] 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.

[16] F. Mannhardt and D. Blinde, "Analyzing the trajectories of patients with sepsis using process mining," in *29th International Conference on Advanced Information Systems Engineering (CAiSE)*, 2017, pp. 72–80.

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

[18] S. J. J. Leemans, E. Poppe, and M. T. Wynn, "Directly follows-based process mining: Exploration & a case study," in *International Conference on Process Mining, ICPM, Aachen, Germany*, 2019, pp. 25–32.

[19] 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," *Inform. Syst.*, vol. 56, 2016.

[20] W. M. P. van der Aalst, D. Tacke genannt Unterberg, V. Denisov, and D. Fahland, "Visualizing token flows using interactive performance spectra," 2020. [Online]. Available: https://www.youtube.com/watch?v=MkBQ_JXyiVs&feature=youtu.be

Discovery of Activities' Actor Perspective from Emails based on Speech Acts Detection

Marwa ELLEUCH [1,2],Oumaima ALAOUI ISMAILI[1],Nassim LAGA[1], Nour ASSY [3], Walid GAALOUL[2]
[1]Orange Labs, France, email : {*FirstName*}.{*LastName*}@*orange.com*
[2] Telecom SudParis, Institut Polytechnique de Paris, France, email: {*FirstName*}.{*LastName*}@*telecom-sudparis.eu*
[3] Lebanese University, Faculty of Information, Beirut, Lebanon, email : {*FirstName*}.{*LastName*}@*ul.edu.lb*

***Abstract*—Process mining aims at discovering different perspectives of business processes (BP) from event logs generated by BP management systems. However, BP can be entirely or partially performed outside such systems. Emails are widely used as an alternative tool to collaboratively perform BP tasks. Recently, there have been several initiatives to extend the scope of BP mining to consider email logs. However, most of them have been mainly focused on discovering BP activity perspective neglecting other equally important knowledge. Actor perspective is one of the important knowledge that can ensure more understanding regarding individuals acting for performing BP activities. Mining such perspective from emails logs provides additional information about the precise contribution of actors in the execution of BP activities. Such information is not limited to executing activities but also refers to requesting, informing, planning or observing activities' execution. This paper first formalizes the knowledge we may discover from emails related to actors perspectives. Then, it introduces an approach based on speech act detection from textual content of emails for discovering such knowledge. Our approach is validated using a public email dataset. Our results are publicly provided to be a first step towards ensuring reproductibility in the studied area.**

***Index Terms*—activity, actor perspective, emails, speech act**

I. Introduction

Process mining consists of discovering BP knowledge from structured event logs. However, some BP or BP parts are not necessarily supported by a BP management system that produces structured events logs. Therefore, applying traditional process mining techniques generates at best partial view of such BP. Emails are widely used as a collaborative tool to support BP execution. However, given the unstructured nature of email data, BP mining techniques cannot be directly applied. Thus, structured event logs need to be generated by extracting BP-related information from unstructured email data. Most existing works have been interested in extracting activities with the aim of mining the BP activity perspective [1]–[6]. Activities' actors perspective (also called organizational or resource perspective) presents one of the relevant BP knowledge that provides additional insights. This perspective basically describes the people assigned to various job functions in a BP [7]. It can reveal the organizational structure, relations and interactions of BP activities' performers [8], [9]. It also allows analyzing the influence of actors on BP performance [10] or mining teams' composition of collaborative activities [11].

Mining actor perspective from emails raises two challenges: (1) What kind of ***additional information*** (oriented actor perspective) that emails can provide comparing to traditional event logs?, and (2) How to ***discover*** them ?

As for the first challenge, traditional event logs are generally generated by BP management systems and contain only the trace of *activities' performers* [8]–[10]. This is limited as several actors can actually intervene, differently, in the achievement of an activity. They would not necessary be performers; they can intervene with different contribution types that would vary from *requesting the execution* of an activity, *requesting information for execution*, *informing about execution*, *observing* how the activity is executed (without doing concrete action as the case of managers for some activities) or simply *executing* the activity. A recent work [11] discovered different actors involved in the execution of an activity. However, it supposed that an activity is represented and implemented as a sub-process and actors are performers of this sub-process. Therefore, interactions with other collaborators (not necessarily tasks' performers) remain missed.

As for the second challenge, only few works have studied activities' actor perspective discovery from emails [12]–[14]. They were mainly based on emails' interlocutors (e.g. senders, receivers) to define activities' actors and weights denoting their involvements. They do not take into account the email content, which provides additional information regarding the contribution of the actors in the activity execution.

To address the aforementioned challenges, this paper proposes an approach to mine the actors' perspective of activities using *speech act information* that can be extracted from emails. It defines the actor perspective as the composition of: (1) the set of actors' groups involved in performing an activity and frequently interchanging activity related emails, (2) the organizational (e.g. manager) and business (e.g. trader) roles of these actors, and (3) actors' contributions (e.g. execution, request, information, etc.) inferred from the detected speech acts of activities' occurrences in emails. Activities in this paper are provided as an input. They are generated using our previous work [15] that introduces a non-supervised approach.

The rest of the paper is organized as follows. Section II gives an overview of the related state of the art. Section III outlines the main abstractions of our approach. Section IV presents the algorithmic solution of some key steps. Finally and before concluding, Section V introduces our evaluation results carried out using the public **Enron dataset**.

978-1-7281-9832-3/20/$31.00 ©2020 IEEE
DOI 10.1109/ICPM49681.2020.00021

II. State Of The Art

Several initiatives proposed to mine BP [1]–[6], [16] and activities [6], [12], [17]–[21] from emails. However, most of them mainly focused on activities' recognition. Only few works targeted the actor perspective of these activities [12]–[14], [21]. They studied it by defining activities' actors and their organizational roles. Some of them [12]–[14] additionally associated to these actors weights denoting their degree of involvement. Two actor involvement diagrams were offered in the visualization tool EmailAnalyzer [14]; (i) task person diagram showing the relation between activities and actors, and (ii) case-person diagram showing the relation between actors and instances. Some of them [13], [14] also studied the social network of activities' related emails' users.
These works [12]–[14], [21] were mainly based on the emails' interlocutors (senders / receivers) or emails' inter-relation (e.g. reply, forward) to uncover actor perspective. Jlailaty et Al [21] have additionally included person names appearing with activities in emails' bodies. Nevertheless, the actual actors' contributions (e.g. request, execution) in performing activities were not defined in these works. Actually, emails contain richer information at the level of their textual content than person names or email addresses. Activities for example are usually introduced in emails with specific speech acts which could give indications concerning actors' contributions.

In the context of discovering activities from emails, there have been existing works that dealt with activity name recognition by using act theory based methods [17]–[20]. Most of them are supervised based approaches; they classify emails or emails' sentences according to senders' speech acts [18]–[20] or use these speech acts as key passage to deduce activities [17]. In the literature, three possible classifications of speech acts are proposed: (1) Illocutionary act classes [22]: Assertive, Directive, Expressive,etc. (2) Speech act verbs [23]: Propose, Request, Deliver,etc. and (3) Emails-oriented speech acts [17]: Salutation, Chit-chat, meeting, promise, farewell,etc. These works have reduced BP activities into a set of speech acts which is not realistic. Actually, activities differ from one BP to another and speech acts would further specify the sender purpose from introducing activities in emails. However, activities' recognition step remains required.

Introduced approaches in this section have considered an email [1]–[6], [13], [16], an email subject [14] or an email sentence [6], [12], [17]–[21] as the lowest structural level that would express: (1) one activity, and/or (2) one sender speech act. Actually, in the context of non-controlled textual data as emails, the expression of these two elements would not be constrained by emails' structure or punctuation. Employees could then express more than one activity instance with different speech acts even in the same sentence. Taking the example of the email retrieved from the Enron dataset (Fig. 1); In the first sentence, the sender **informs** recipient(s) about one activity instance of opening a long trading position. As for the last sentence, the sender introduces three activities instances with two different speech acts; (i) cut a deal (colored by grey)

We are short 100 mw's in the Ercot Asset book for hours ending 7-22. We want to remain that way, unless the balancing energy market goes haywire.
We would need to delete our SC counterparty El Paso and input a replacement QSE in the portal if this is filled.
Enpower shows 100 mw's coming from Ercot imbalance

This deal would need to be cut and replaced with the power that was purchased for bal-day, etc.

Fig. 1. Email sample from Enron Dataset

and replace a deal (framed by dashed rectangle) in the form of **intention** , and (ii) purchase energy (framed by continuous line) in the form of **information about execution**.

III. Approach Main Notations

Our approach discovers activities' actor perspective from emails. Due to the unstructured nature of emails log data in the context of activities, our approach needs to transform it into a structured format. Such format is an event log that records useful information at each occurrence of an activity in an email. This section formalizes then the input and the output of our approach and the concept of our structured *event log*.

A. Input: Emails & Activities

1) Emails Log: Emails log (Definition 1) in our input is of semi structured format comparing to its raw format. In fact, email parts (e.g. main body) and relations between emails (e.g. reply) are supposed to be detected. Emails' interlocutors roles in a company are also supposed provided which are of two types; (i) Organizational role (OR) (e.g. manager), and (ii) Business role (BR) (e.g. trader). In case of external interlocutors (not employees), roles are denoted as 'external'.

Definition** 1: **(Emails Log Data: EmL) An emails log data (EmL) is a set of emails where each email $em = (id, tm, sender, to, cc, subj, mainbdy, cvhist, next_{ids})$ has nine elements: a unique identifier (id), a timestamp (tm), a sender ($sender$), list of receivers with different status (to, cc), a subject ($subj$), a body composed of a main body ($mainbdy$) and a conversational history ($cvhist$) and finally, a set of ids ($next_{ids}$) of emails sent as reply or forward of em and having timestamps higher than tm. Each $u = (add, or, br) \in to \cup cc$ has an email address (add), an OR (or) and a BR (br).

2) Activities: Activities in this paper are provided by our previous work [15] which analyses emails to discover the most occurring activities (in an unsupervised way). An activity (Definition 2) was defined as the composition of: (1) **Activity Name (AN)** that reflects its main goal (e.g; 'create deal'), and (2) A set of **Business Data (BD)** that refer to the data used or generated during its execution (e.g; {'deal price', 'ID'}). Let $\mathcal{AN}$ and $\mathcal{BD}$ denote the sets of AN and BD.

Definition** 2: **(Activity) An activity is defined as $Act = (AN, BD)$ such that: (1) $AN \in \mathcal{AN}$, and (2) $BD = \{bd_1, ..bd_p\} \in \mathcal{BD}^*$. $\mathcal{A}$ will denote the set of all activities.

To detect activities in emails without sentences' structure constraints (as discussed with Fig. 1), the same previous work maps each activity component type (AN or BD) to a set of patterns used by employees to express it (Definition 3). **A pattern** was defined as a set of lemmatized words associated with their syntactic functions (e.g. verb, noun). Let $\mathcal{W}$, $\mathcal{T}$ and $\mathcal{P}$ denote

the sets of words, syntactic tags and patterns. If $p \in \mathcal{P}$, it is defined as $p = \{(w_1, t_1), ..(w_l, t_l)\}$ where $(w_j, t_j) \in \mathcal{W} \times \mathcal{T}$ $\forall j$. Taking the example of the activity name AN_1 ='purchase power' of Fig. 1, p_1={('purchase','v'),('power','n')} and p_2={('buy','v'),('numeric','n'),('mw','n')} are among the patterns mapped to it where 'v' ('n') refers to verb (noun) syntactic tag. Patterns provide then a semantic characterization so that an activity can be detected in emails even when using different words: (1) having synonymy relation (e.g. 'buy'/'purchase'), or (2) sharing the same business context of use; p_2 allows the detection of AN_1 by referring the purchased quantity ('numeric) of power in megawatt ('mw') rather than explicitly mentioning the word 'power'. An **AN pattern** must contain at least one verb. **A BD pattern** does not contain verbs and must contain tags of numeric values (e.g. price) or of named entities' types (e.g. location, person name).

Definition 3: (Mapping-To-Patterns) A Mapping-To-Patterns function is defined as $\mathcal{M}_{Pat} : \mathcal{AN} \cup \mathcal{BD} \rightarrow \mathcal{P}^*$ where $\mathcal{M}_{Pat}$ returns the list of patterns associated to an activity component type ($\in \mathcal{AN} \cup \mathcal{BD}$).

B. Output: Actor Perspective

In the context of emails, for each activity, an actor is a user present in the interlocutors' lists of emails containing it. Let $\mathcal{AT}$ be the set of activities' actors. This paper defines an actor perspective of an activity according to Definition 5 which leverages defining the notion of its actors' groups (Definition 4). An actor group of an activity refers to the set of actors that coexist frequently (according to th_{min}) in the same interlocutors lists of emails related to the same activity.

Definition 4: (Activity Actors' Group) Let $Act \in \mathcal{A}$ be an activity. $G_{Act} = \{a_1, ..a_m\} \subset \mathcal{AT}$ is a group of actors of $Act \Leftrightarrow \forall (a_i, a_j) \in G_{Act} \times G_{Act}, \frac{|E_i \bigcap E_j|}{min(|E_i|, |E_j|)} > th_{min}$, while; (1) $th_{min} \in [0, 1]$, (2) $E_k = f_{em}(a_k, EmL, act)$ for $k \in \{i, j\}$, and (3) $f_{em} : \mathcal{AT} \times EmL \times \mathcal{A} \rightarrow EmL$ is the function that returns for each actor ($\in \mathcal{AT}$) the set of emails (related to Act) while he belongs to their interlocutors lists.

Definition 5: (Activity Actor Perspective) Let $Act \in \mathcal{A}$ be an activity. Let $\mathcal{G}_{Act}$, be the set of unique actors groups' ids of Act. Let $\mathcal{OR}$ and $\mathcal{BR}$ denote the sets of OR and BR of actors in the company. Actor perspective of an activity Act is defined as a tuple $\Omega_{Act} = (Ar, \mathcal{F}, R, C)$ while;
- $Ar = \{a_1, .., a_n\} \subset \mathcal{AT}$ is the set of n actors;
- $\mathcal{F} : Ar \rightarrow \mathcal{G}^*_{act}$ is the function that maps each actor to his groups' ids (one actor can be affected to multiple groups);
- $R = \{(or_1, br_1), .., (o_n, b_n)\} \subset \mathcal{OR} \times \mathcal{BR}$ is the set of actors' roles such that or_i and br_i are the OR and BR of a_i;
- $C = \{\sigma_1, \ldots, \sigma_n\}$ is the set of distributions of actors' contributions towards the activity act. Each distribution $\sigma_i = \{(c, \lambda_{c,i}) \mid c \in \zeta \wedge \lambda_{c,i} \in [0, 1]\}$ refers to the distribution of the contributions ($c \in \zeta$) of actor a_i towards the activity Act such that $\sum_c \lambda_{c,i} = 1$. A tuple $(c, \lambda_{c,i}) \in C_i$ refers to a contribution c and its fraction $\lambda_{c,i}$;

Six contribution types $\in \zeta$ are identified based on our analysis of industrial emails and discussions with BP experts; (1) **Request** the execution of an activity from other actors, (2) **Request Information** concerning the execution of an activity (e.g., status, useful data, opinion, permission), (3) **Execution** by performing the activity, (4) **Information** about the execution of an activity (executed or not), (5) **Planning** by expressing intentions of executing an activity (even by other actors), and (6) **Observation** of how an activity is performed, without doing a concrete action towards its execution.

C. Event Logs

Compared to the standard event log format provided by **XES** standard, our event log (Definition 9) differs in three ways. First, the notion of BP instance (trace) is omitted since it does not play an important role in discovering our actor perspective. Second, the notion of *activity occurrence* (Definition 7) is introduced as an alternative to the well-know concept of *activity instance* in process mining. The main difference is that activity occurrence refers to the appearance of its AN (and optionally of its BD) in an email without; (1) necessarily being executed, or (2) disposing precise information concerning its instantiating. Third, the structure of our event logs must allow the recording of information referring to actors' contributions at each activity occurrence which are related to the notions of; (1) *Activity Component occurrence* (Definition 6), and (2) *Speech Act* (Definition 8) of activity occurrence. In what follows, let $\mathcal{E}$ denote the set of email main bodies (after transforming each one into a format similar to $p \in \mathcal{P}$).

Definition 6: **(Activity Component Occurrence)** Let $AC \in \mathcal{AN} \cup \mathcal{BD}$ be an activity component. Let $Pos : \mathcal{P} \times \mathcal{E} \rightarrow \mathbb{N}^*$ return the positions' list of appearance of words' pattern $p \in \mathcal{P}$ in an email $e \in \mathcal{E}$ sorted in ascending order. $AC_{occ} = (AC, R_f)$ is the occurrence of AC in e $\Leftrightarrow$ (1) $\exists p \in \mathcal{M}_{pat}(AC)$ while $w \in e$ $\forall w \in p$ and $\forall i \in [1, |Pos(p, e)| - 1]$, $|Pos(p, e)[i + 1] - Pos(p, e)[i]| \leq th_d \mid th_d \in \mathbb{N}$, AND (2) R_f is the raw form of patterns' words in e (before lemmatization).

An activity component type occurs then in an email e if one of its patterns is detected while its words are low dispersed. Low dispersity of words (ensured by a defined threshold th_d) means that they appear close to each other in e. This guarantees that if a pattern is detected in e, it will express the same activity component objective (as it was supposed in [15]). Considering; (1) an email extract e_A = 'I have created deal ticket 241558 for July', and (2) an activity $Act_2 = (AN_2, \{bd_2\})$ where $p_{an} = \{('create', 'v'), ('ticket', 'n')\} \in \mathcal{M}_{pat}(AN_2)$ and $p_{bd} = \{('deal', 'n'), ('numeric', 'n')\} \in \mathcal{M}_{pat}(bd_2)$. In case of p_{an} (p_{bd}), the words *'create'* and *'ticket'* (*'deal'* and *'numeric'* tag reflecting '241558') appear close to each other in e_A inducing the detection of AN_2 (bd_2) occurrence where; $AN_{2occ} = (AN_2, \{'created', 'ticket'\})$ ($bd_{2occ} = (bd_2, \{'deal', '241558'\})$). In what follows, let $\mathcal{O}$ denote the set of activities' components' occurrences.

Definition 7: **(Activity Occurrence)** Let $Act = (AN, BD) \in \mathcal{A}$ be an activity. Let $f_{occ} : \mathcal{AN} \cup \mathcal{BD} \times \mathcal{E} \rightarrow \mathcal{O}^*$ be a function that returns the occurrences of an activity component in an email $e \in \mathcal{E}$. Act has an occurrence

$Act_o = (AN_{occ}, BD_{occ}) \in \mathcal{O} \times \mathcal{O}^*$ in $e \Leftrightarrow$ (1) $AN_{occ} \in f_{occ}(AN, e) \neq \emptyset$, AND (2) $BD_{occ} = \{f_{occ}(bd, e), bd \in BD\}$.

An activity is considered then detected in an email if mainly a pattern representing the activity name is detected (because BD values would not always be present in the same email).

***Definition** 8:* **(Speech Act)** Let $Act_o = (AN_{occ}, BD_{occ}) \in \mathcal{O} \times \mathcal{O}^*$ an occurring activity. A speech Act (SA) of Act_o defines the purpose behind introducing AN_{occ} in an email e.

In this paper, four speech acts (SA) types are considered:
- **Request act:** The actor requests the execution of an activity from the recipient(s), e.g. 'can you schedule an interview with this candidate?' for the activity 'schedule interview'.
- **Request information act:** The actor requests some information concerning the execution of an activity (status, useful data), e.g. "who is the best person in london to forward the resume to?" for the activity 'forward resume'
- **Intention act:** The actor expresses an intention of doing the activity in the future (by himself or other actors), e.g. "kate, my assistant, shirley crenshaw, will schedule a meeting."
- **Information act:** The sender uses the email for informing about activity execution status (it was executed or not or in current execution), e.g. the activity 'create deal' of e_A.

***Definition** 9:* **(Event Logs)** An event log (EL) is a set of events while each event is defined as $ev = (id, Act_o, ActorInd, SA, em)$ where: $Act_o = (AN_{occ}, BD_{occ}) \in \mathcal{O} \times \mathcal{O}^*$ is the occurring activity; $ActorInd$ is a list of textual indicators that may refer to the activity real doer (e.g. personnel pronoun, person or organization name); SA is the activity speech act; and $em \in EmL$ is the email in which the activity occurred.

IV. Approach

Our approach is composed of three main phases (Fig. 2):

Phase1: Preprocessing: This phase preprocesses each email main body. It removes some stopwords and useless words (e.g.; thanking, salutations, signatures). It detects also some numeric values and named entities and it replaces them by tags (e.g., pricenumeric, personname). Finally, it returns, a structured format (a tuple $t = (w, pos, w_b, t_s, sent)$) allowing the detection of activities' patterns while recording information concerning; (1) words' raw form (w, e.g. "created" of email extract A), (2) its position of appearance (pos) in the email, (3) words' basic or lemmatized form (w_b), e.g. "create" is the basic form of "created", (4) words' syntactic function tags (t_s): only two types of tags are considered: verb (v) and non-verb (n) as they are important in defining activity components, and (5) the sentences ($sent$) where words appears in the email.

Phase2: Event logs entities detection and generation from emails logs (detailed in Section IV-A): It mainly detects events' attributes (according to Definition 9) present in emails' main bodies. It first *detect activities' occurrences (Act_o)* using their characterization in term of patterns (provided by our previous work [15]). For each Act_o, it records its exact position of appearance in an email and the sentence where it appeared. Then, it exploits these two information to find; (i) *indicators of actor* involved in the occurring activities ($ActorInd$), and (ii) SA of activities occurrences.

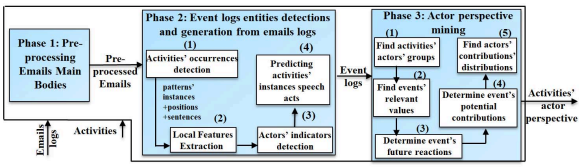


Fig. 2. Approach Main Steps

Phase3: Actor perspective mining from event logs: (detailed in Section IV-B) It analyses the obtained event logs to discover, for each activity, its actor perspective. Related actors' contribution are defined: (1) directly by deducing them from speech acts of activity occurrences, or (2) indirectly by considering actors' indicators or tracking actors reactions towards received emails.

A. Phase2: Event logs entities detection from emails logs:

1) Step1: Activities' occurrences detection: This step detects the occurrences of activities in emails using their related patterns. It aims to: (i) find their exact positions of appearance in emails, and (ii) extract the sentences where they occurred.

For each activity component $AC \in AN \cup BD$ having a set of patterns $P = \mathcal{M}_{pat}(AC)$, this step is mainly based on iteratively searching all the occurrences of each pattern ($\in P\}$) in each email while verifying the low dispersity criterion. First, it replaces words' pattern in an email by tags reflecting them. The syntactic functions of these words in the email must match those of the pattern. After that, the email will be reduced to the list of pattern's tags while keeping their real position of appearance in the email (without removing redundancy). The elements of this list are then sorted in an ascending order according to these positions. By scrolling through this list, each element will be added to the same pattern occurrence if: (i) its tag was not previously added, (ii) the position distance that separates it from the last added element does not exceed a defined threshold (th_d), and (iii) it is assigned to the same sentence of the tag previously added. Once having an element where (ii) or (iii) is not verified, this latter will mark the end of one potential pattern occurrence and the beginning of another. Examples illustrating the results of this step are in this link[1].

2) Step2: Local features extraction: It uses the entire sentence where an activity name occurred to extract local features (about its verb) useful for next steps and which are of two types; (i) grammatical, and (ii) verb voice; passive or active. Grammatical features are obtained after **parsing grammatical dependencies** [24] from the sentence (without removing standard stop words as they would be useful). This returns a tuple of three additional elements comparing to t:
- **Words' POS tag**: A POS (Part-Of-Speech) tag of a word indicates some grammatical categories such as word's tense or number (plural/singular), e.g. past participial, verb ING;
- **Words' Dependency tags**: each word's tag defines its grammatical relation with the verbal element to which it depends, e.g. 'subject' ('agent') is a dependency tag between

the action's performer and verb in active (passive) voice;
- Words' Subtrees: A subtree of a word w refers the the subset of words having direct grammatical dependencies with w, e.g. {'that', 'was', 'purchased', 'for', 'bal-day'} forms the subtree of 'purchased' in the last sentence of Fig. 1.

For identifying, for each activity occurrence, **the voice of the verb** being used to express it, this step uses verb's subtree to check if it verifies the specific grammatical structure of a passive voice sentence. Otherwise, the verb voice is considered as active. In this paper, this specific structure is considered as the composition of: (i) an auxiliary dependency tag referring to an auxiliary verb ('be' or sometimes 'get'), and (ii) A past participle POS tag of the main verb denoting the action.

3) Step3: Extracting actor indications: These indicators would be useful to define the actual contribution of some emails' interlocutors. In fact, if the sender uses the first (or the second) personnel pronoun to express the doer of the activity action, it is an indication that he (or some recipients) is (are) the actual executor(s). This step studies two cases:
(i) If the verb of the activity is in passive voice: the doer of the action may be specified in what follows the activity verb using a prepositional phrase with the preposition 'by' e.g. "it was purchased yesterday by Mark". This steps localizes then such prepositional phrase in case it was indicated.
(ii) If the verb of the activity is in active voice: the doer of the action may be specified by the subject of the sentence subtree of the activity verb. The position of such subject is then determined in order to find its correspondent textual content.

4) Step 4: Predicting activities' occurrences' speech acts: For predicting SA once an activity occurs, two methods are proposed; (i) **rules based method**: It is an unsupervised method that defines a decision tree summarizing some rules derived form natural language, and (ii) **supervised method**: It applies any existing supervised learning algorithm. It does not require a human expertise to define prediction rules. However, it leverages a labeled training dataset.

These two methods are based on analysing four kinds of properties of each verb characterizing an activity occurrence;
- *Grammatical properties:* they concern the verb's voice, form (in basic form or not), POS tag and dependency tag .
- *Subtree properties:* they concern words (basic form & grammatical properties) preceding the verb in its subtree.
- *Neighbourhood properties:* they concern words preceding the verb and appearing close to it in the same sentence. The closeness of a word is defined by a distance threshold separating its position of appearance from that of the verb.
- *Sentence properties:* They concern some specific words (e.g. 'I', 'we', 'you', 'please' ..) or punctuation ('?') appearing in the overall sentence of the verb.

In case of **the supervised method**, for each occurred activity, learning features are deduced from POS tags and dependency tags of the defined properties' verbs. They include also some specific words indicating tense (e.g. yesterday), personnel pronoun (e.g. 'I', 'we', 'you'), request words (e.g. 'please') and sentence punctuation. As they must have numerical representations during the learning phase, this method calculates the distance that separates them from the activity verb (in term of absolute value of difference between their positions of appearance and that of the verb). In case of non existence of one feature, a large distance value is considered.

In case of **Rules based method**, it aims to define SA on the basis of activity verb tense (that could be deduced from verb POS tag). In fact, if emails' senders use activity verb in past, they would inform about its execution (e.g. 'I created a ticket'). If they use it in future, they would express an intention of execution. As for verb in present or in basic form, additional properties (e.g. neighbourhood) must be analysed to identify the correspondent SA. This method defines then a set of rules to ensure tense-SA mapping and summarizes it in the form of a decision tree (Fig. 3). Our tree is composed of ten levels (discontinuous rectangles, e.g. L1, L2). Each level could contain; (i) a diamond shape referring to a verb property to be verified, e.g. D1: 'Verb voice is active' (ii) Rounded shape indicating tree's end terminators while precising the predicted SA, e.g. SA=Request, and (iii) Rectangle shape referring to an action to be done after a decision, e.g. A2: Return to D1. Levels of our tree are of three categories:
- *Sentence form decision levels (L1&L2):* they concern (i) sentence punctuation (L1), if it is a question, SA will be a request information act unless the verb is preceded by a specific modal verb (MD1, e.g. can, could, would) to be considered as a request act, and (ii) verb voice (L2), if it is in passive voice, SA is assigned to an information act unless the auxiliary verb is not expressed in its basic form (i.e 'be'). Otherwise, the verb is replaced by its auxiliary verb..
- *Own and neighborhood decision levels (L3 → L6):* Starting from these levels, the verb is considered in an active voice and some own and neighbourhood properties are checked. The closer they are to the verb, the higher the priority of the corresponding rules/decisions. As verb POS tag is an own property, it has a higher priority. This tag could directly indicate if the verb is in 'Past' tense or not to recommend if SA is an information act (L3). Then, the priority pass to some specific words appearing in the verb neighborhood; (i) modal

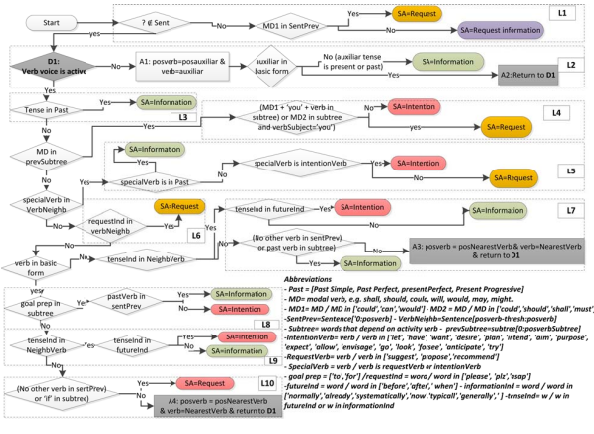


Fig. 3. Speech Acts Prediction Tree [1]

verbs in L4 ('MD', e.g. will, would, shall) recommending that SA is an intention act unless 'you' is present in verb subtree as a subject or switched with some specific modal verbs (MD2, e.g. can, would), (ii) some special verbs (in L5) of intention (e.g. want, plan) or of request (e.g. suggest) recommending that SA is an intention or request act unless they are expressed in 'Past' (SA will be information), and (iii) some special words indicating request SA ('requestInd', e.g. 'please') in L6.

- *Transitive decision levels ($L7 \rightarrow 10$):* Here, verb is considered either (i) third-person verb, or (ii) in basic form ($L8 \rightarrow L10$). Verb neighborhood would be not sufficient to recommend tense or SA. Some properties are then checked to transfer the SA prediction to the closer verb preceding the activity verb. This operation will be repeated unless no verb precedes the activity verb or SA is predicted. If at the end, SA is not yet predicted, SA will be assigned to information (L7) and request (L10) in case of (i) and (ii) respectively.

B. Phase3: Actor perspective mining from event logs

This phase has as input the event log (EL) generated from the second phase. For each activity type of name AN, it generates a sublog by projecting EL on AN by keeping only events with activity occurrence name AN. Then it uses this sublog to discovers its activity actor perspective. Algorithm 1 illustrates the main operations applied on each sublog:

Step1: Cluster actors by groups (Line 2,3): It uses the coexistence fraction, as defined in Definition 4, to group sets of actors frequently interchanging activity related emails.

Step2: Define relevant values of activity occurrences (Line 4): It characterizes one activity occurrence by values enabling their tracking in future emails (Ideally by unique values). Two types of data are considered: (1) receivers' email addresses (of external users, e.g. candidates) and (2) BD (e.g. deal number). These data types are considered relevant if their values are largely distinct, which means the fraction of unique values among all values is close to 1 (as each one tends to be unique).

Step3: Determine future reactions of each event (Line 7): It tracks future emails sent by actors to identify event related emails. These emails can be (i) of 'reply' or 'forward' relation with the event's email, (ii) of bodies containing an event relevant value, or (iii) sent to relevant event's email addresses.

Step4: Determine potential actors' contributions of each event: This step analyses the attributes of each event to estimate the potential contributions of its related email's interlocutors. These contributions are: (1) directly deduced from event SA (through $Map[Ev.SA]$); information, request, request information and intention acts refer to information, request, request information and planning contributions made by email sender (Line 8), and (2) indirectly deduced from event SA by exploiting the values of the other event attributes; a potential contribution of execution is assigned to: (a) email sender if SA is information and 'I' belongs to event's actor indicators (Lines 10,11), (b) email receiver (Lines 12→17) if; (i) SA is request, or SA is information and 'you' belongs to event's actor indicators and (ii) email is of unique receiver of sate 'To', or the receiver belongs to the list of senders of future reactions' emails (only in the case of request SA).

Step5: Determine the distribution of actors' contributions: This step performs some operations of aggregation on the overall emails' interlocutors and their potential contributions. Fist, actors of observation contribution are identified if they are always receivers with 'CC' Status without having execution contribution (lines $19, 20$). Then, contribution affectations of Step4 are grouped by actors. Finally, for each actor i, $\lambda_{c,i}$ ($C[act][c]$) of each contribution (c) will be the fraction of its number of occurrences in the trace of affectations (line 21).

Algorithm 1 Activity SubLog Mining

Input: $SL(SubLog), EmL(EmailLogs), th_{min}$
Output: Ar, G, R, C

1: $contrb\_act = \{P : [], Ex : [], R : [], Obs : [], RI : [], I : []\}$
2: $TOs = [ev.em.to\ for\ ev\ in\ SL\ ]$, $CCs = [ev.em.cc\ for\ ev\ in\ SL]$, $senders = [ev.em.sender\ for\ ev\ in\ SL]$
3: $Ar = TOs \cup CCs \cup senders$, $G = FindActorsGroups(SL, Ar, th_{min})$
4: $RelevBDVal, RelevAddr = LargeVar(SL)$
5: **for** Ev **in** SL **do**
6: tos=$Ev.em.to$,ccs=$Ev.em.cc$, $sender = Ev.em.sender$
7: $futureReacts = SearchReacts\ (Ev.em,\ EmL,\ RelevBDVal,\ RelevAddr)$
8: $contrb\_act[Map[Ev.SA]].append(sender)$
9: **if** $Ev.SA == 'informationact'$ **then**
10: **if** $'i'$ **in** $Ev.ActorInd$ **then**
11: $contrb\_act[Ex].append(sender)$
12: **if** $'you'$ **in** $Ev.ActorsInd$ & $len(tos) == 1$ **then**
13: $contrb\_act[Ex].append(tos[0])$
14: **if** $(Ev.SA == 'requestact')$ & $len(tos) == 1$ **then**
15: $contrb\_act[Ex].append(tos[0])$
16: **for** r **in** tos **do**
17: **if** $r\ in\ futureReacts.senders$ & $Ev.SA == 'requestact'$ **then**
18: $contrb\_act[Ex].append(r)$
19: $Ctr\_Rec = Counter(TOs \cup CCs)$, $Ctr\_CCs = Counter(CCs)$
20: $contrb\_users[Obs] = [\ r\ for\ r\ in\ Ctr\_CCs.keys()\ if\ \frac{Ctr\_CCs[r]}{Ctr\_Rec[r]} > 0.5\ \&\ r \notin contrb\_act[Ex]]$
21: **for** act **in** A **do**
22: $affcs$ = $FindAff(contrb\_act[Obs], act)$, $ctr = Counter(affcs)$
23: $C[act][c]$ = $\frac{ctr[c]}{len(affectations)}$ **for** c **in** ζ =0

V. Experiments and Evaluation Results

This section evaluates our approach on real emails belonging to Enron dataset. The evaluation is split into two parts. One part evaluates a key step which is *SA prediction of activities' occurrences* to justify our choice among the two proposed methods. The second evaluates *event logs generation* and *actor perspective mining* phases. It studies, per activity sublog, the accuracy of occurrences detection and SA prediction (of the

TABLE I
Evaluation Dataset

Ep	*OR*	*BR*	$\mathbf{Lab_{viz}}$	$\mathbf{N_e}$	$\mathbf{N_{re}}$	$\mathbf{N_a}$
E1	Managing Director	Trading	MD_Tr0	343	421	106
E2	Senior Counsel	Legal	SC_Lg0	102		
E3	Managing Director	Risk	MD_RM1	2283	504	
E4	Assistant	Management	$Asst\_3$	357		
E5	Manager	Logistics	M_Log1	738	682	
E6	Specialist	Settlements	$Spec\_sett1$	108		
E7	Specialist	Logistics	$Spec\_Log19$	100		
E8	Employee	Employee	$E203$	158		
E9	Specialist	Logistics	$Spec\_Log21$	80		

selected method). It highlights also our visualization for an activity actor perspective. Our approach was implemented using ***Python*** and some **Natural Language Processing libraries**, e.g. *Spacy* to lemmatize words and detect their subtrees and their syntactic, POS and dependency tags.

For discovering activities using our previous work [15], we used emails (of Enron) sent by nine employees to different collaborators. Then, activities' patterns were used to detect activities' occurrences (with $th_d = 3$) in emails sent by (1) the same set of nine employees, and (2) employees that frequently receive activities' oriented emails. Business and organizational roles (BR, OR) of employees were annotated on the basis of either their email signatures or some public resources (e.g. investigation report of enron). In case of their unavailability, they were denoted as "employee". Table I summarizes the features of our evaluation dataset. For each employee, BR and OR are indicated. N_e and N_{re} are the number of emails (of non empty main body) sent by him and by his selected receivers. Lab_{viz} is his visualization label (in Fig. 4). N_a indicates the number of all detected activities (occurring more than 7 times).

A. Speech acts prediction

This part evaluates our SA prediction methods. First, a sample of 937 activities' occurrences were selected from our evaluation dataset (mainly from emails of E1, E3 and E4). Then, each activity occurrence was annotated according to its SA. Finally, *rules based method* was applied on the overall annotated data. As for *the supervised method*, data were split into training data (70%) and test data (30%). Some supervised learning algorithms were then tested; SVM (Support Vector Machine), RF (Random Forest) and DT (Decision Tree).
To compare the predicted SA with the annotated ones, F1-score was used. The obtained results are summarised in Table II. For each SA, the total of annotated data (N) is indicated with the correspondent F1-scores of our rules based method ($f1_r$), SVM ($f1_{SVM}$), RF($f1_{RF}$) and DT ($f1_{DT}$). As noticed, the two variant of our SA prediction method tend to have good scores ($\in [0.79, 0.91]$). As rules based method obtained the higher score, its results were adopted for the remained steps.

B. Event Log Generation and Actor Perspective Mining

This part evaluates our event log and shows the practical relevance of the actor perspective mining step. The generated event log contains 2359 events and 106 activities (i.e 106 sublogs). Details and sublogs (in form of csv files) of the most occurred activities are provided in this link [1]. Due to lack of space, table III summarizes only those of the 10 most occurred activities. Each sublog is identified by an ID (e.g. $el2$), the correspondent activity (e.g. *'create_make deal'*), the number of events N_{ev} (e.g. 139) and the number N_{gr} of the detected groups of actor (with $th_{min} = 0.5$). A_O and A_{SA} refer to the accuracies of; (1) occurring activities detection, and (2) SA prediction. They reflect their variation according to emails' senders through this format: $WF \pm sd$ where; WF is the weighted average of accuracies per sender and sd is their standard deviation. The obtained results show good performances in term of accuracies. They show also how they can be affected by; (1) activity, e.g. 0.12 difference between A_O of $el10$ and $el1$) and (2) email's sender (e.g. $sd = 0.14$ for $el10$) which is due to a variation in writing style.

One of the main motivations of discovering activities' actors in BP is to analyze the underlying social network. Since our main novelty is the discovery of actors' contributions, a visualization tool that takes them into account was implemented using *networkx* and *matplotlib*. Our implementation provides, for each activity, three different visualizations as illustrated in Fig.4 (which shows the actor perspective of *'el2'*): two graphs

[1]http://www-inf.it-sudparis.eu/SIMBAD/tools/ActivityActorPerspective/

TABLE II
F1-SCORES

SA	***N***	$f1_r$	$f1_{SVM}$	$f1_{RF}$	$f1_{DT}$
intention	416	0.92	0.8	0.82	0.8
information	333	0.91	0.82	0.82	0.8
request	132	0.9	0.77	0.8	0.78
request information	56	0.9	0.91	0.85	0.77
Total	**937**	**0.91**	**0.8**	**0.81**	**0.79**

TABLE III
SUBLOGS OF MOST 10 OCCURRED ACTIVITIES

ID	***Activity***	N_{ev}	A_O	A_{SA}	N_{gr}
el1	sell_trade numeric0mw{gas}	144	0.99 ± 0.015	0.92 ± 0.045	6
el2	create_make deal	139	0.94 ± 0.073	0.95 ± 0.08	2
el3	enter deal	126	1 ± 0	0.9 ± 0.016	2
el4	change_convert deal	102	0.93 ± 0.05	0.93 ± 0.06	4
el5	purchase_buy numeric0mw{gas}	96	0.95 ± 0.05	0.9 ± 0.045	5
el6	set_adjust deal	77	0.95 ± 0.06	0.84 ± 0.09	7
el7	extend deal{numeric}	67	1 ± 0	0.8 ± 0.06	2
el8	conduct_bring interview	66	0.98 ± 0.04	0.96 ± 0.01	5
el9	send_attach resume	60	0.98 ± 0.015	0.96 ± 0.03	4
el10	see_check deal_trade	49	0.87 ± 0.14	0.85 ± 0.14	3

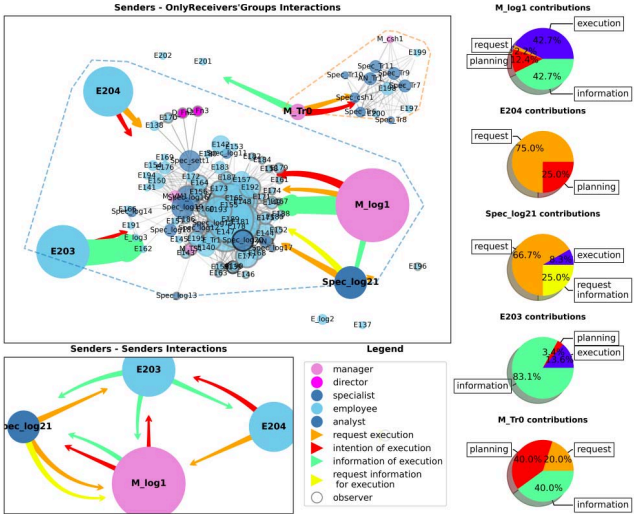


Fig. 4. Create Deal Actor Perspective

visualizing the interactions between senders and receivers, and a pie chart showing the overall contributions of senders.

Nodes in the graphs refer to the actors of the activity. These actors are of two types: (1) senders: that frequently send emails about the activity (more than 5 times in case of Fig. 4). They could also be receivers, and (2) OnlyReceivers: that receive emails about the activity. These actors may react to the emails sent by "senders" by replying or forwarding emails but do never send activity related emails. Nodes' size, color and label reflect; (i) the number of emails concerning the activity that was sent or received, (ii) the actor OR, and (iii) concatenation of the actor OR, BR and an anonymous ID. The interactions represented by edges are colored according to the SA detected in the email. Their width refers to the number of emails concerning the activity that was sent or received.

1) Senders-OnlyReceivers interactions: This graph shows the interactions made by senders and OnlyReceivers. These interactions (i.e. edges) are of two types; (i) Undirected edges indicating that two actors were receivers of the same email (they are not connected in case one of them is a sender). A Connected group of nodes refers to a group of OnlyReceivers receiving the same emails; (ii) Directed edges referring to a number of emails sent from a sender to a group of OnlyReceivers with specific SA. Each closed dashed line in the graph regroups actors of the same group, e.g. blue and orange lines.

2) Senders-Senders interactions: It summarizes interactions between senders, e.g. 'E203' always send information of execution to other senders while receiving intentions or requests.

3) Senders' Pie-chart contributions: it illustrates the fractions in (%) characterizing the distribution of each sender's contributions. e.g. E204: 75% request & 25% planning.

Our visualization outlines the different interactions and contributions made by actors of the same activity (per BR or/and OR). It would provide more understanding regarding activity execution rights. Given the example of the employees *Spec_log21* and *E204*, it's clear that they do not have the right of creating a deal (through an information system). This is why they always contact a manager (*M_log1*) or a specialist (*E203*) having access. This would recommend; (1) who to be contacted for executing an activity or for intervening in case of a delayed activity, and (2) extensions to be implemented in an information system to further automate actors' interactions.

VI. Conclusion

This paper proposed an approach for discovering activities' actor perspective from emails. It defines an actor perspective considering the additional information that would be present in emails and it proposes a related visualization. It differentiates from existing works by defining actors' contributions in performing an activity. To this end, it was mainly based on SA detection of activities occurrences. Two variants were proposed for predicting SA; rules based method and supervised method. Our experiments were carried out on a public dataset and the results are publicly shared (link[1]), which is, in our knowledge, absent in related works (This is why comparison with them was not feasible when evaluating our proposals).

The obtained results show good performances with emails of the selected employees. We agree that further analysis (on larger set of employees) must be carried out to study if our main assumptions remain valid while varying writing styles. These assumptions concern mainly our rules used for SA prediction and the relevance of the selected features.

This work presents a preliminary step towards mining actor perspectives of the overall BP that can be discovered from emails. Towards this goal, we are currently investigating how to take advantage of existing relevant metrics (e.g. work handover, activities reassignment) and extend them by adding richer information that can be mined from email data.

References

[1] D. Jlailaty, D. Grigori, and K. Belhajjame, "Mining business process activities from email logs," in *IEEE ICCC*, pp. 112–119, 2017.

[2] C. Di Ciccio, M. Mecella, M. Scannapieco, D. Zardetto, and T. Catarci, "Mailofmine–analyzing mail messages for mining artful collaborative processes," in *SIMPDA*, pp. 55–81, 2011.

[3] C. Di Ciccio and M. Mecella, "Minerful, a mining algorithm for declarative process constraints in mailofmine," *Department of Computer and System Sciences Antonio Ruberti Technical Reports*, 2012.

[4] N. Laga, M. Elleuch, W. Gaaloul, and O. AlaouiIsmaili, "Emails analysis for business process discovery," *ATAED*, p. 54, 2019.

[5] R. Banziger, A. Basukoski, and T. Chaussalet, "Discovering Business Processes in CRM Systems by leveraging unstructured text data," in *HPCC*, pp. 1571–1577, 2018.

[6] N. Kushmerick, T. Lau, M. Dredze, and R. Khoussainov, "Activity-centric email: A machine learning approach," in *AAAI*, p. 1634, 2006.

[7] B. Curtis, M. Kellner, and J. Over, "Process modeling. communications of acm," vol. 35, pp. 75–90, 1992.

[8] W. M. Van Der Aalst, H. A. Reijers, and M. Song, "Discovering social networks from event logs," *CSCW*, vol. 14, no. 6, pp. 549–593, 2005.

[9] M. Song and W. M. Van der Aalst, "Towards comprehensive support for organizational mining," *Decision support systems*, vol. 46, no. 1, pp. 300–317, 2008.

[10] J. Nakatumba and W. M. van der Aalst, "Analyzing resource behavior using process mining," in *BPM*, pp. 69–80, Springer, 2009.

[11] S. Schönig, C. Cabanillas, C. Di Ciccio, S. Jablonski, and J. Mendling, "Mining team compositions for collaborative work in business processes," *SoSyM*, vol. 17, no. 2, pp. 675–693, 2018.

[12] D. C. Soares, F. M. Santoro, and F. A. Baião, "Discovering collaborative knowledge-intensive processes through e-mail mining," *Journal of Network and Computer Applications*, pp. 1451–1465, 2013.

[13] T. M. Mitchell, S. H. Wang, Y. Huang, and A. Cheyer, "Extracting knowledge about users' activities from raw workstation," in *AAAI*, vol. 1, p. 181, 2006.

[14] W. M. van der Aalst and A. Nikolov, "Emailanalyzer: an e-mail mining plug-in for the prom framework," *BPM Center Report*, 2007.

[15] M. Elleuch, O. Alaoui Ismaili, N. Laga, W. Gaaloul, and B. Benatallah, "Discovering activities from emails based on pattern discovery approach," in *BPM Forum*, 2020.

[16] N. Kushmerick and T. Lau, "Automated email activity management: an unsupervised learning approach," in *IUI*, pp. 67–74, 2005.

[17] E. K. Ringger, R. Campbell, S. Corston-Oliver, and M. Gamon, "Task-focused summarization of email," 2004.

[18] W. W. Cohen, V. R. Carvalho, and T. M. Mitchell, "Learning to classify email into "speech acts"," in *EMNLP*, pp. 309–316, 2004.

[19] M. Jeong, C.-Y. Lin, and G. G. Lee, "Semi-supervised speech act recognition in emails and forums," in *EMNLP*, pp. 1250–1259, 2009.

[20] A. Qadir and E. Riloff, "Classifying sentences as speech acts in message board posts," in *EMNLP*, pp. 748–758, 2011.

[21] D. Jlailaty, D. Grigori, and K. Belhajjame, "On the elicitation and annotation of business activities based on emails," in *ACM/SIGAPP*, pp. 101–103, 2019.

[22] J. R. Searle, "A taxonomy of illocutionary acts," 1975.

[23] J. R. Searle and J. R. Searle, *Speech acts: An essay in the philosophy of language*, vol. 626. 1969.

[24] M. Honnibal and M. Johnson, "An improved non-monotonic transition system for dependency parsing," in *EMNLP*, pp. 1373–1378, 2015.

Process Mining over Unordered Event Streams

Ahmed Awad
Cairo University, Giza, Egypt
University of Tartu, Tartu, Estonia
ahmed.awad@ut.ee

Matthias Weidlich
Humboldt-Universität zu Berlin, Berlin, Germany
matthias.weidlich@hu-berlin.de

Sherif Sakr
University of Tartu, Tartu, Estonia
sherif.sakr@ut.ee

***Abstract*—Process mining is no longer limited to the one-off analysis of static event logs extracted from a single enterprise system. Rather, process mining may strive for immediate insights based on streams of events that are continuously generated by diverse information systems. This requires online algorithms that, instead of keeping the whole history of event data, work incrementally and update analysis results upon the arrival of new events. While such online algorithms have been proposed for several process mining tasks, from discovery through conformance checking to time prediction, they all assume that an event stream is ordered, meaning that the order of event generation coincides with their arrival at the analysis engine. Yet, once events are emitted by independent, distributed systems, this assumption may not hold true, which compromises analysis accuracy.**

In this paper, we provide the first contribution towards handling unordered event streams in process mining. Specifically, we formalize the notion of out-of-order arrival of events, where an online analysis algorithm needs to process events in an order different from their generation. Using directly-follows-graphs as a basic model for many process mining tasks, we provide two approaches to handle such unorderedness, either through buffering or speculative processing. Our experiments with synthetic and real-life event data show that these techniques help mitigate the accuracy loss induced by unordered streams.

I. Introduction

Traditional process mining takes place in a postmortem fashion. An event log is extracted from an enterprise system, which is then analyzed. Specific analysis tasks, for instance, include the discovery of a process model [1], the assessment of conformance of the log with a given model [10], or the construction of a model for time and outcome prediction [13].

Recently, the application scenarios of process mining have been greatly extended. This includes scenarios in which streams of event data from diverse information systems are used to provide near-real time operational insights and decision support. In such a streaming setting, events arrive continuously, at potentially high rates, from various systems. Hence, analysis has to rely on online algorithms that store solely a limited amount of data and work incrementally. That is, each event is processed only once, upon arrival, and incorporated by updating the analysis results. Respective online algorithms have been proposed not only for process discovery [8], [18], but also for conformance checking [28], and prediction tasks [22].

Once event streams are emitted by independent, distributed systems, the order in which the events are generated (the so-called 'event time') does not necessarily coincide with the order in which the events arrive at a process mining engine (the 'arrival time'). That is, out-of-order arrivals of events may be observed [20]: Events arrive for processing *before* all older events have arrived. Such a situation, referred to as *unorderedness* of the stream, is common for distributed event streams, due to synchronization issues, network delays, and intermittent connectivity. Assuming that events for a business process are collected from information systems, e.g., a Web portal, an ERP system, a manufacturing system, and an SCM system, the events arriving at a process mining engine cannot be expected to be ordered by their event time, even if each system emits its events in-order. Incorporating sensed event data for analysis, e.g., from real-time locating systems in logistics or health care [25], renders this problem even more prominent.

Existing online algorithms for process mining, whether for discovery, conformance checking, or prediction, assume that an event stream is ordered, i.e., the arrival order corresponds to the event time order. When these algorithms are applied to unordered streams, erroneous or at least inaccurate results are obtained. The reason being that most techniques exploit control-flow information, so that the ordering of events is of utmost importance for the analysis. Specifically, a directly-follows graph (DFG), derived for a single case or a set thereof, captures which activities have been executed directly after one another. It provides a basic model employed by many contemporary process discovery algorithms, see [1], and is used to define behavioral features in the construction of prediction models [19]. Considering a DFG per case, the captured order is also used in alignment-based conformance checking [10].

To illustrate the effect of out-of-order arrivals, consider the events in Table I. Here, the 'Timestamp' refers to the event time, while the 'Arrival TS' denotes the arrival time. Figure 1 further visualizes different scenarios of event arrivals for the first three events of case 1, ordered versus unordered. The lower part also illustrates that an online algorithm to construct the DFG solely based on the arrival order will yield erroneous results, suggesting that a sequence A,C,B of activity executions was observed. Handling unorderedness explicitly, an online algorithm shall yield the same result as if all events would have arrived in-order.

In this paper, we provide techniques to handle unordered event streams in process mining. We focus on the computation of a DFG as a basic model for many process mining tasks and show how to achieve resilience against unorderedness. Specifically, we make the following contributions. We formalize the notion of unordered event streams and discuss the applicability of a number of established strategies to handle it

978-1-7281-9832-3/20/$31.00 ©2020 IEEE
DOI 10.1109/ICPM49681.2020.00022

TABLE I
RUNNING EXAMPLE OF AN UNORDERED EVENT STREAM.

Case ID	Activity	Timestamp	Arrival TS
1	A	14:00	14:10
1	B	14:30	14:50
1	C	14:40	14:45
2	A	15:00	15:31
2	B	15:17	15:27
2	D	16:32	17:32
1	E	17:00	17:05
2	C	17:05	17:15
2	E	17:07	17:14

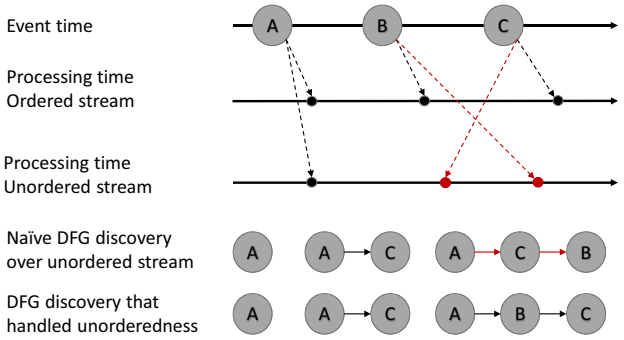


Fig. 1. Effect of unorderedness on the DFG computation.

in the context of process mining. Based thereon, we propose a pipeline-based model for incremental computation of the DFG and introduce two operators to handle unorderedness that employ either buffering or speculative processing. The respective design choice is reviewed regarding the advantages and disadvantages of either operator.

We implemented our approach on top of Apache Flink, a framework for stream data processing, and conducted experiments with synthetic and real-life event data. Our results indicate that buffering and speculative processing help mitigate the accuracy loss induced by unordered streams.

The rest of this paper is organized as follows. Background information is presented in Section II, which is followed by a discussion of related work in Section III. Our approach to process mining over unordered event streams is introduced in Section IV. Our implementation and experiments are detailed in Section V, before we conclude in Section VI.

II. BACKGROUND

A. Event, Traces and Logs

Event logs record the execution of a process. The recording of a single instance of a process (a.k.a. case) is a *trace*, a finite sequence of *events*. An event, in turn, is a manifestation of the execution of an activity of the process. An event records important information about the execution of the activity, such as the case and the time it was executed. It may include further information, e.g., on the (human) resource that executed the activity [27]. In the context of this paper, we require events to contain at least a case identifier, an activity, and a timestamp, which correspond to the first three columns in Table I.

In traditional *postmortem* process mining, the order among events is derived from their timestamps. Turning to *online* process mining, however, events are processed in the order of their arrival at a process mining engine. As illustrated above, the differences in the order of event generation as manifested in the timestamps ('event time') and as derived from the event arrival ('arrival time') have severe implications for the analysis.

Definition 1 (Event): Let C be the set of all case identifiers, A be the set of all activities, and T be the (ordered) domain of timestamps. An event e is tuple $(a, c, t_e, t_a) \in A \times C \times T \times T$, where t_e is the *event time* and t_a is the *arrival time*.

We refer to the properties of an event using dot notation. For example, for an event e, $e.a$ refers to the activity and $e.t_e$ refers to its event time. For our work, it is useful to consider streams of events per case, rather than a single, global stream.

Definition 2 (Event stream): Let E be the set of all events. An event stream is an unbounded sequence $\sigma = \langle e_1, e_2, \dots \rangle$, such that $e_x.c = e_y.c$ and $e_x.t_a < e_y.t_a$ for any $1 \leq x < y$ and $e_x, e_y \in E$. We write $e_x \in \sigma$ to indicate that event e_x is part of stream σ.

In Definition 2, we enforce the order among the events based on their arrival time. When an event stream corresponds to a completed case, it is called a trace. An event stream is said to be *ordered*, when the event time of arriving events grows monotonically. Otherwise, the stream is called *unordered*.

Definition 3 (Unordered event stream): Let σ be an event stream. It is unordered, if there exists $e_x, e_y \in \sigma$, such that $e_x.t_a < e_y.t_a$ and $e_x.t_e > e_y.t_e$.

For illustration, consider the following two streams constructed using the events from Table I and their *Timestamp*[1] and *Arrival TS* columns respectively, for the first three events. Then, stream $\sigma_1 = \langle (A, 1, 14{:}00,\ 14{:}00), (B, 1, 14{:}20, 14{:}30), (C, 1, 14{:}40, 14{:}40) \rangle$ is ordered, whereas the second stream $\sigma_2 = \langle (A, 1, 14{:}00, 14{:}10), (C, 1, 14{:}40, 14{:}45), (B, 1, 14{:}30, 14{:}50) \rangle$ is unordered.

Given the events of one or more cases, a directly-follows graph (DFG) may be derived (also known as a process map) [8], [18]. It represents a basic model in process mining that captures which activities have been executed directly after one another.

Definition 4 (Directly follows graph): Let A be the set of activities. A directly-follows graph is a tuple $DFG = (V, E, w)$ where $V \subseteq A$, $E \subseteq A \times A$ and w is a function that assigns weights to the arcs, $w : E \to \mathbb{N}_0 \cup \{-1\}$.

Unlike traditional notions of a DFG, the above definition allows for a negative weight -1 for edges. Later, we will use these negative weights to *retract* dependencies between activities due to unorderedness of an event stream.

In online process mining, a DFG is constructed incrementally. With the arrival of new events, nodes and edges can be added, and weights of existing edges can be altered.

Figure 2 illustrates the evolution of the DFG for the unordered stream σ_2 introduced above. In Figures 2(a) and 2(b), new nodes and edges with positive weights are added to the

[1]Assuming that events are received according to their event time.

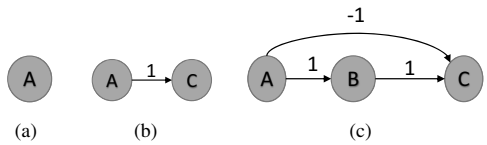


Fig. 2. Incremental maintenance of the DFG for stream σ_2: (a) after arrival of event A; (b) after arrival of C; and (c) after the out-of-order arrival of B.

DFG, since the arrival times coincides with the event times. Event B then arrives out-of-order, since its event time is smaller than the one of event C. Hence, the weight of edge (A, C) is set to -1, indicating that the edge has to be retracted. Additionally, the edges (A, B) and (B, C) are added with weight 1.

B. Techniques to Handle Unordered Streams

In order to yield accurate results, online process mining has to account for unordered event streams and handle out-of-order arrivals of events. In the general field of data stream processing, techniques to handle unorderedness can be classified as buffer-based, punctuation-based, or speculation-based [12].

Buffer-based techniques delay the processing of stream elements for a slack period θ [3]. Once this time has passed, elements are sorted and passed to the downstream operator for processing. Out-of-order arrivals beyond θ are ignored.

Punctuation-based techniques depend on a special stream element that indicates the progress of the stream. As such, buffering is applied, but not for a fixed time period, but based on the streaming data. That is, elements are buffered until a punctuation is received that indicates that all elements up to a certain time points have arrived.

Speculation-based processing is optimistic [5] and assumes that no out-of-order arrivals will occur. Hence, results of a computation are emitted immediately with the arrival of new elements. If late elements arrive, the emitted result is invalidated and an update that incorporates the late element is emitted. Thus, the downstream operator has to employ a sophisticated logic to react to result updates. Speculation-based techniques incur a significant overhead compared to buffering when event streams show frequent late arrivals. However, they yield low processing latency and are generally more resilient to unorderedness than buffering. To balance the latency and computational effort, a slack period θ may also be applied on top of a speculative technique [23].

In the context of process mining, we will focus on buffer-based and speculation-based techniques. The reason for not considering punctuations is that they cannot be expected to be present in event streams in the general case. Punctuations represent an assumption on the capabilities of the systems generating events, which would limit the applicability of process mining techniques.

III. Related Work

Shifting the focus of process mining from static event logs to streams of event data, a number of *online* algorithms have been proposed in recent years. In general, such online techniques need to address the following requirements, or at least a subset thereof, see also [26]: *R1*, they shall build and maintain analysis results incrementally; *R2*, they shall access each received event only once, or for a limited number of times; *R3*, they shall provide analysis results with low latency; *R4*, they shall adapt to fluctuating event arrival rates; *R5*, they shall handle stream imperfections, such as unorderedness.

Focusing on process discovery, Kindler et al. [16], [17] proposed an incremental procedure that works on batches of completed cases. From each batch, a process model, a Petri net, is constructed. Merging these models, a global Petri net that covers the behavior observed across multiple batches is derived, thereby addressing requirements *R1* and *R2*. We note that the ability to merge partial results is at the very core of any approach to incremental maintenance. However, in [16], [17], this merging step is actually not detailed.

Burattin et al. [8] propose to conduct process discovery over a sliding window and through lossy counting with budget. The window is used to collect a *finite* subset of recent events (i.e., events become outdated at some point) from the stream to update the discovered model, whereas lossy counting is adopted for the frequencies of the directly-follow relation. Hence, the approach covers requirements *R1-R3*, while *R4* has been addressed through an implementation on a scalable architecture in [4]. In [18], a similar approach is proposed that addresses memory limitations (the size of the discovered model), by applying and comparing cache management techniques.

Other work proposes the use of sequential pattern mining to discover continuously evolving process models from event streams [15]. Similar to the approaches above, this approach employs a forgetting mechanism for outdated events, and a decaying mechanism, to stay within memory limits. Again, this covers requirements *R1-R3*.

The approaches discussed thus far are geared toward discovery of imperative process models. Yet, similar approach have also been proposed to discover declarative process models from event streams [7].The work addresses requirements *R1-R4*.

Turning to conformance checking between an event stream and a model, it was proposed to rely on a decomposition of a process model along with a tailored token replay technique [29], which covers requirements *R1-R3*. Moreover, alignment-based techniques to conformance checking have been lifted to a streaming setting. In [28], prefix alignments are introduced to handle events as they arrive (requirements *R1* and *R2*). Yet, when computing accurate results instead of approximations, these techniques may suffer from performance issues. An alternative angle for online conformance checking have been put forward in [6], which relies on pre-computed possible deviations, and in [9], [30], which follow the idea to derive behavioral patterns the process model for online evaluation. Again, requirements *R1-R3* are addressed by these approaches.

However, all of the discussed approaches assume that the event stream is ordered. Hence, requirement *R5* related to stream imperfections is not addressed. In this paper, we aim to close this gap with a generic technique to handle unorderedness, a specific type of stream imperfection. By

focusing on DFGs as a basic model, our technique can be used to achieve robustness against unorderedness for most of the aforementioned algorithms. Thus, providing a complementary approach that addresses *R5*.

IV. Handling Unorderedness in Process Mining

This section introduces our approach to handle unorderedness for online algorithms to process mining. We take up the ideas of buffer-based and speculation-based techniques for unorderedness in the general field of stream processing, see Section II-B. Unlike punctuation-based techniques, these approaches do not enforce assumptions on the systems generating events.

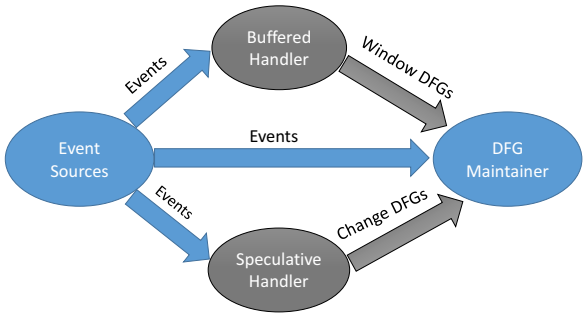


Fig. 3. Three alternative pipelines for online process mining.

In general, stream processing jobs can be described adopting a data flow programming model [24]. That is, events are processed through a pipeline that is captured by a directed acyclic graph (DAG). Nodes in the graph are operators that apply some transformation to their input data, while the edges in the graph show how the transformed results are used as input to further operators. A source operator is responsible for receiving a stream of raw data and feeding it into the pipeline. Symmetrically, a sink operator receives the results of the transformations and forwards it for storage and interpretation.

For the context of process mining that is based on an online construction of a DFG, Figure 3 illustrates three pipelines. First, events may directly flow from the *event sources* to the *DFG Maintainer* operator, which incrementally constructs the DFG and also serves as a sink. Yet, as detailed above, this approach will potentially lead to erroneous results for unordered streams. We therefore suggest to consider two alternative pipelines, in which another operator, a *Buffered Handler* or a *Speculative Handler*, is injected before the incremental computation of the DFG. Either of these two operators and, hence, the respective pipelines strikes different trade-offs, as will be discussed in the remainder.

For illustration, we will take up the event stream of Table I. For this stream, the final DFG derived based on the timestamps (i.e., the event times) is given in Figure 4.

A. Buffered Handler

A buffered handler has a slack parameter θ, which can be seen as a time window for which the operator keeps receiving events from the stream. As the window expires, the operator reorders the events by their timestamps (event time) (Definition 1). Then, the reordered events can be sent to the DFG maintainer. Yet, to help distributing the computation, one may also first construct one or more window DFGs of these events, which are subsequently merged by the DFG maintainer with a global DFG that is updated continuously.

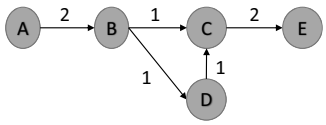


Fig. 4. Final DFG after processing all events of Table I in-order.

Once the results have been sent downstream, the operator resets, keeping solely the most recent event w.r.t the timestamp. Any future event with a timestamp less than the one of this event is ignored. Hence, out-of-order events may arrive *too late* in relation to the slack time θ. A too small value of θ will cause inaccuracies in the DFG computation, whereas a too large value of θ will yield significant delays before the DFG is updated. However, adaptive techniques can be used to dynamically change the value of θ at runtime [11], [2].

When the operator produces its output as a stream of *ordered* events, the DFG maintainer works as in the traditional setting. This can be seen as a 1:1 transformation: For every event received by the buffered handler, there will be a corresponding output event. Only in the case of a *too* late arrival, the event will be discarded. When the operator produces its output as a DFG, the DFG maintainer operator is slightly modified to *merge* the received window DFG with the so-far constructed global DFG. This merging is additive, meaning that the global DFG will only be extended with nodes or edges, or the weights of existing edges are incremented:

Definition 5 (DFG merge): Let $g_1 = (V_1, E_1, w_1)$ and $g_2 = (V_2, E_2, w_2)$ be two DFGs. Function $merge(g_1, g_2)$ produces a new DFG $g' = (V', E', w')$ where $V' = V_1 \cup V_2$, $E' = E_1 \cup E_2$ and $\forall\ e \in E', w'(e) = \begin{cases} w_1(e) + w_2(e) & e \in E_1 \wedge e \in E_2 \\ w_1(e) & e \in E_1 \wedge e \notin E_2 \\ w_2(e) & e \in E_2 \wedge e \notin E_1 \end{cases}$

To produce correct results, the buffered handler needs to keep a buffer of events for each case separately.

Figure 5 illustrates how a buffered handler works on the unordered event stream of Table I. We set the slack time $\theta = 30$ minutes. Then, we have time windows, such as the one [14:00-14:30).[2] Note that the window start time is inclusive and the end time is exclusive. Any event with an arrival timestamp greater than or equal to 14:00 and less than 14:30 will be included in the window [14:00-14:30).

In the first window, only the event of activity A in case 1 is received. This event will be kept as the most recent event for case 1. The DFG generated for this window just contains the event for activity A and this will also be the global DFG.

[2]In Table I, we show only a snapshot of the windows relevant to the arrival timestamps of the event stream.

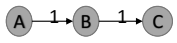
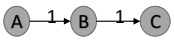
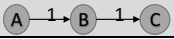
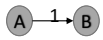
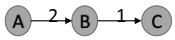
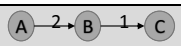
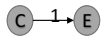
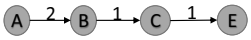
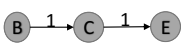
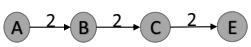
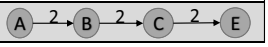
Time window	Content	Latest element	Window DFG	Global DFG
[14:00 – 14:30)	(1, A, 14:00, 14:10),	(1, A, 14:00, 14:10)		
[14:30 – 15:00)	(1, C, 14:40, 14:45), (1, B, 14:30, 14:50)	(1, C, 14:40, 14:45)		
[15:00 – 15:30)				
[15:30 – 16:00)	(2, B, 15:17, 15:27), (2, A, 15:00, 15:31)	(2, B, 15:17, 15:27)		
[16:00 – 16:30)				
[16:30 – 17:00)				
[17:00 – 17:30)	(1, E, 17:00, 17:05)	(1, E, 17:00, 17:05)		
	(2, E, 17:07, 17:14), (2, C, 17:05, 17:15)	(2, E, 17:07, 17:14)		
[17:30 – 18:00)	(2, D, 16:32, 17:32)	(2, E, 17:07, 17:14)		

Fig. 5. Buffered handling of event stream of Table I.

In the second window, events of activities B and C arrive out-of-order. Yet, they are captured as their timestamps are greater than the latest element seen previously for this case. To generate the window DFG, these events are ordered based on their timestamps. With the latest element from the previous window, the graph corresponds the sequence of the events of A, B and C. The window DFG graph is sent to the DFG maintainer to be merged with the current global DFG (Definition 5). The window [15:00-15:30) does not contain any events.

Events of case 2 start to arrive in window [15:30-16:00). Although the events arrive out-of-order, they belong to the same window. The window DFG is constructed and the merge with the global DFG causes the update of the weight of edge (A, B). In window [17:00-17:30) events from two cases arrive and they are maintained in two different buffers. For case 1, the most recent event from the previous window is related to C. In this window, an event for activity E is received, so that the window DFG defines the relation between E and C. For case 2, events C and E arrive out-of-order and are reordered. Then, the window DFG is constructed, where the most recent event from the previous window for case 2 is that of activity B. We also show the global DFG after merging the two window DFGs. The most recent event for case 2 is that of activity E.

In window [17:30-18:00), the event of activity D from case 2 arrives. Yet, since its timestamp is less than that of the most recent event of activity E. Hence, the event of activity D is ignored (highlighted in red in Figure 5). As a result, no window DFG is constructed and the global DFG is not updated.

Comparing the last global DFG in Figure 5 with the one in Figure 4, there is a difference: D is not present in the last global DFG and also the weights of the edges are different. We will formalize the measurement of the difference in Section V.

B. Speculative Handler

The speculative handler provides updates for the computation of the DFG upon the reception of new events. In particular, out-of-order arrivals in an unordered stream may lead to updates that retract edges of the DFG. In comparison to the buffered handler, the speculative handler tends to maintain a larger state. The reason being that the buffered handler only needs to keep track of the most recent event, w.r.t to the timestamp, for each case. The speculative handler, in turn, needs to maintain the whole DFG for each case.

Figure 6 shows how the speculative handler processes the event stream of Table I. With the arrival of the event of activity B of case 1, the operator identifies the out-of-order arrival by a comparison of the events' timestamps. Based thereon, a change DFG is generated, which consists of three nodes and three edges. The insertion of the late event at the right position requires us to invalidate the directly-follows edges between the preceding and the succeeding events. In our case, these are the events of activity A and activity C for case 1, respectively. As a result, the change DFG contains the edge (A, C) with weight -1 (Definition 4) and two edges (A, B) and (B, C), with weight 1, reflecting the directly-follows dependencies induced by the received event. The DFG maintainer operator will merge this change DFG with the current global DFG to obtain a new global DFG (Definition 5). For the case of the out-of-order arrival of the third event in the stream, the edge (A, C) in the global DFG will have weight 0 and, thus, is removed. The same effect happens when events related to activities C and D for case 2 arrive out-of-order. Incorporating all the respective change DFGs, the global DFG is identical to the one obtained under in-order arrival (Figure 4).

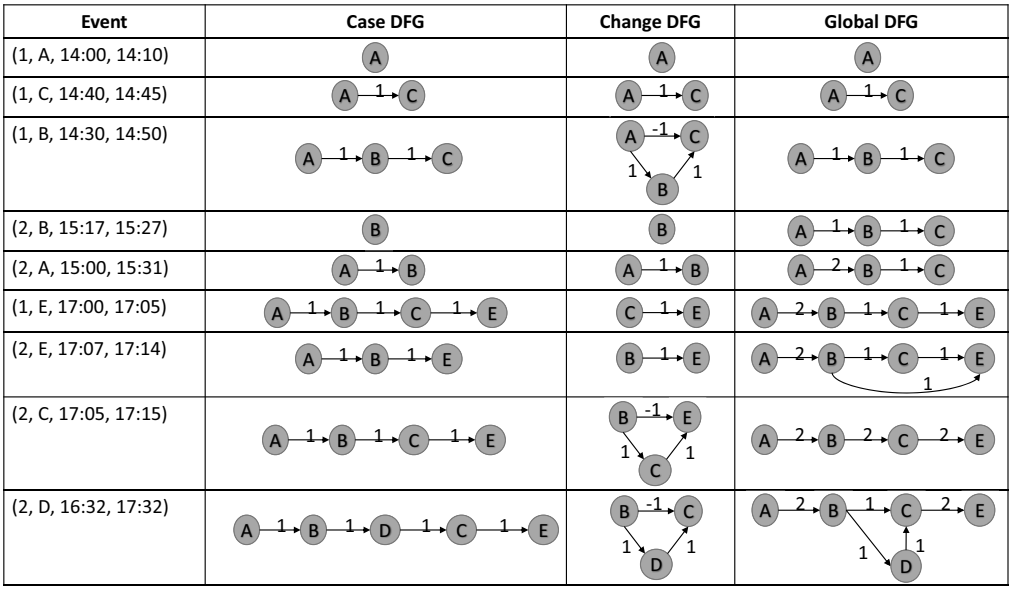


Fig. 6. Speculative handling for the event stream of Table I.

V. Evaluation

A. Implementation

We have implemented the two operators on top of Apache Flink,[3] a scalable stream processing engine. For the buffered handler, we have used the built-in `event time window` and have provided a custom mechanism to trigger the computation. Most recent events are tracked with a custom state object to store the latest event seen on the stream (Section IV-A). For the speculative handler, we used the low-level `Process Function`, which is triggered for each arriving stream element. Again, a custom state object was used to maintain the prefixes of the traces to derive the change DFGs (Section IV-B). Our implementation is available in a Github repository.[4]

B. Experiments

Datasets. We evaluated our approach using both synthetic and real-life event data. The former was derived using a model previously proposed to assess the performance of algorithms for concept drift detection [21]. For this model, event data was generated using the BIMP simulator.[5] We generated events for a total of 250 cases, each comprising 10 events on average. Case durations varied from 1 to 10 hours.

To introduce unorderedness, we relied on the out-of-order generator [14]. The tool enabled us to control the percentage of events that will arrive out-of-order, as well as the minimum and the maximum delay for events. In our experiments, we have set the minimum delay to 3 hours and the maximum delay to 10 hours. For the percentage of delayed events, we varied the value from 10% to 90% at 10% increments.

[3] https://flink.apache.org/

[4] https://github.com/DataSystemsGroupUT/Process-Discovery-over-unordered-streams

[5] http://bimp.cs.ut.ee/simulator

Regarding real-life data, we considered the BPI challenge dataset of 2015.[6] It covers processes of a Dutch municipality. For our experiments, we took subset of the data for cases that occurred in 2014. The timestamp of events is recorded at the granularity of days, so that many events have the same timestamp. Yet, there is another event attribute, which points to the *planned* completion time of the activities. We used this attribute as the timestamp to derive an unordered stream and, through sorting by this attribute, also an ordered stream.

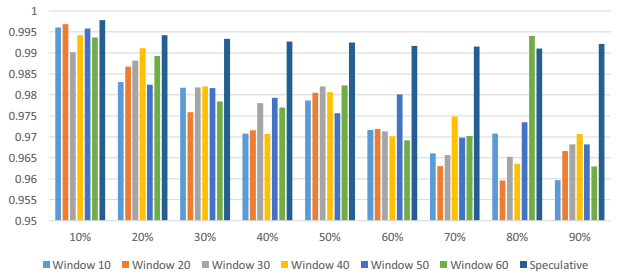
Setup and procedure. We ran the three pipelines of Figure 3 on a local Flink cluster. First, we fed the ordered events to the buffered handler, the speculative handler, and directly to the DFG maintainer, respectively. Then, we recorded the evolution of the global DFG to serve as the baseline. Next, each unordered stream (with a certain percentage of out-of-order events) was fed into the three pipelines and, again, we recorded the evolution of the global DFG.

For the buffered handler, we varied the slack time θ from 10 minutes to 60 minutes at steps of 10 minutes. When comparing the results to the traditional pipeline that does not handle unorderedness, however, we made sure that the latter also uses a buffer to make the results comparable.

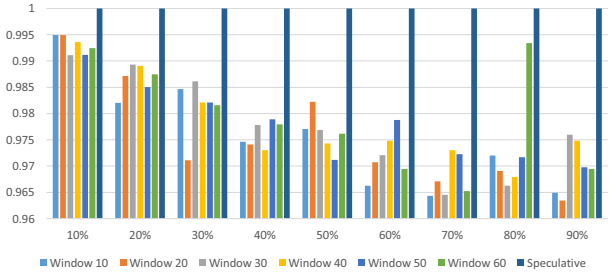
Metrics. To measure the accuracy of the generated DFG, we compared the global DFG after each trigger of the respective mechanisms, and compare the results with the baseline. That is, after each window had been processed by the buffered handler, we compared the global DFGs from the unordered and the ordered stream. Likewise, for the speculative handler, DFGs where compared after processing each element. We used the following metric to compare DFGs:

$$accuracy(g, g') = \frac{2 - \frac{loss}{totalfrequency_g} - \frac{excess}{totalfrequency_{g'}}}{2} \quad (1)$$

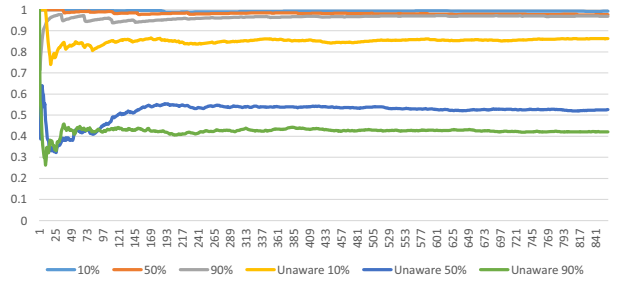
[6] https://www.win.tue.nl/bpi/doku.php?id=2015:challenge



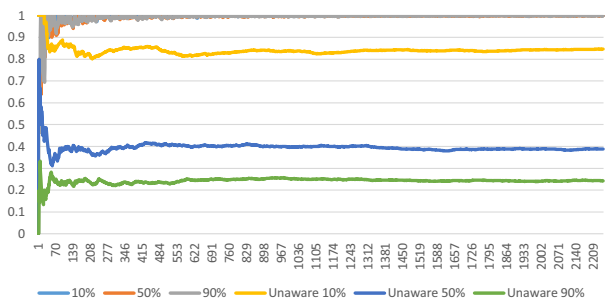
(a) Average accuracy. X-axis: out-of-order percentage



(b) Accuracy of the last global DFG. X-axis: out-of-order percentage.



(c) Buffering over a window of 60 minutes. X-axi:s number of windows.



(d) Speculative processing. X-axis: number of events processed.

Fig. 7. Results for the synthetic dataset. Y-axis shows accuracy

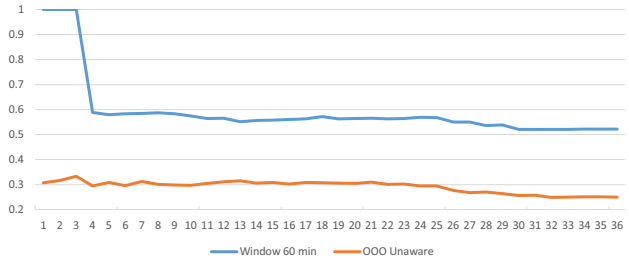
Here, *loss* refers to edges and their weights that are in the baseline DFG g, but *not* in g' that evolves from the unordered stream. Conversely, *excess* sums the weights of edges in g' not present in g, due to out-of-order arrivals. The obtained accuracy value is between 0 (no matches at all) and 1 (identical graphs).

Results. We first focus on the average accuracy (Figure 7(a)) and the accuracy obtained after processing the whole data (Figure 7(b)) for both the buffered and the speculative techniques for the synthetic log. We notice some variance of the accuracy within each group of streams with a certain percentage of unorderedness related to the window size. Also, with increased unorderedness, the accuracy obtained with the buffered technique is more affected compared to the speculative approach, especially in Figure 7(b). When comparing the DFGs once the last window (buffered handler) or last event (speculative handler) has been processed, we notice that the speculative approach is able to yield a DFG that is identical to the one of the baseline (ordered stream). The buffered approach, in turn, leads to a small (the accuracy is still above 0.96 in all scenarios), yet notable difference in the resulting DFGs.

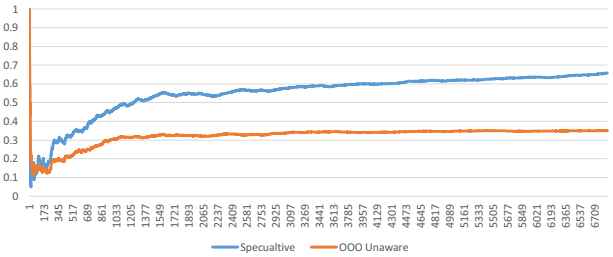
Next, we focus on comparing the two proposed techniques to the case where the unordered stream would have been processed without handling unorderedness explicitly, which corresponds to the middle pipeline in Figure 3. In Figure 7(c), the buffered handler for a window of 60 minutes is compared to the traditional pipeline (*unaware*) under different degrees of unorderedness (i.e., 10%, 50%, and 90% delayed events). Here, the buffered technique is able to maintain accuracy above 0.9, whereas the best accuracy of the traditional approach is about 0.85 for 10% delayed events. The accuracy drops significantly, to about 0.42, as the severity of the unorderedness increases. For the case of speculative processing in Figure 7(d), the accuracy of the speculative approach is converging to 1, whereas the traditional pipeline performs much worse. For a highly unordered stream, the accuracy drops below 0.2.

Figure 8 reports the accuracy of the different techniques over the BPI 2015 dataset. In both cases, the handling of stream unorderedness as proposed in this paper contributes to increased accuracy, which confirms the results obtained for the synthetic dataset. We further notice that speculative processing does not reach an accuracy value of 1 by the end of processing. The reason is that many events (belonging to different activities in the same case) in the dataset have the exact same timestamp. So, when the data is sorted by the timestamp (to form the baseline) the order of two events does not necessarily correspond to the order in the original dataset. This causes observing different edges in the respective DFGs. This phenomenon also contributes to the comparatively lower accuracy of the buffered approach.

From the experimental results, we conclude that, as expected, the speculative approach maintains better accuracy compared to the buffered approach. By definition, it also yields a lower latency, since events are processed immediately upon their arrival. However, this comes at the expense of resources used to maintain the state of evolving cases. An interesting direction for future work, therefore, is to study a hybrid approach that combines buffered and speculative processing, and evaluate its performance from both points of view, the achieved accuracy and the induced resource consumption.



(a) Buffering over a window of 60 minutes. X-axis number of windows.



(b) Speculative processing. X-axis number of events.

Fig. 8. Results for the BPI 2015 dataset.

VI. Conclusion

In this paper, we pointed out the importance of accounting for stream *unorderedness*, when performing online process mining. We presented two approaches to handle out-of-order arrivals of events. That is, a buffered or speculative handler shall be introduced in a pipeline for process mining that builds upon the computation of a directly-follows graph. We have implemented and evaluated the approaches on both, synthetic and real-world datasets. Our results show how inaccuracies due to unorderedness can be mitigated by our techniques.

In future work, we intend to develop a hybrid approach to strike a balance between the desired accuracy level to maintain, and the latency of reporting updates in the DFG computation. Moreover, we intend to lift our techniques to process perspectives beyond control-flow, such as information on temporal dependencies and involved resources.

References

[1] 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. *IEEE Trans. on Knowledge and Data Engineering*, 31(4):686–705, 2019.

[2] A. Awad, J. Traub, and S. Sakr. Adaptive watermarks: A concept drift-based approach for predicting event-time progress in data streams. In *EDBT 2019*, pp. 622–625. OpenProceedings.org, 2019.

[3] S. Babu, U. Srivastava, and J. Widom. Exploiting k-constraints to reduce memory overhead in continuous queries over data streams. *ACM Trans. Database Syst.*, 29(3):545–580, 2004.

[4] A. Batyuk and V. Voityshyn. Streaming process discovery for lambda architecture-based process monitoring platform. In *CSIT*, vol. 1, pp. 298–301, 2018.

[5] A. Brito, C. Fetzer, H. Sturzrehm, and P. Felber. Speculative out-of-order event processing with software transaction memory. In *DEBS*, pp. 265–275. ACM, 2008.

[6] A. Burattin and J. Carmona. A framework for online conformance checking. In BPM Workshops, *LNBIP* 308, pp. 165–177. Springer, 2017.

[7] A. Burattin, M. Cimitile, F. M. Maggi, and A. Sperduti. Online discovery of declarative process models from event streams. *IEEE Trans. Serv. Comput.*, 8(6):833–846, 2015.

[8] A. Burattin, A. Sperduti, and W. M. P. van der Aalst. Control-flow discovery from event streams. In *IEEE CEC*, pp. 2420–2427, 2014.

[9] A. Burattin, S. J. van Zelst, A. Armas-Cervantes, B. F. van Dongen, and J. Carmona. Online conformance checking using behavioural patterns. In BPM, *LNCS* 11080, pp. 250–267. Springer, 2018.

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

[11] T. Das, Y. Zhong, I. Stoica, and S. Shenker. Adaptive stream processing using dynamic batch sizing. In *SoCC*, pp. 16:1–16:13. ACM, 2014.

[12] M. Dayarathna and S. Perera. Recent advancements in event processing. *ACM Comput. Surv.*, 51(2):33:1–33:36, 2018.

[13] C. D. Francescomarino. Predictive business process monitoring. In *Encyclopedia of Big Data Technologies*. Springer, 2019.

[14] P. M. Grulich, J. Traub, S. Breß, A. Katsifodimos, V. Markl, and T. Rabl. Generating reproducible out-of-order data streams. In *DEBS*, pp. 256–257. ACM, 2019.

[15] M. Hassani, S. Siccha, F. Richter, and T. Seidl. Efficient process discovery from event streams using sequential pattern mining. In *Symposium Series on Computational Intelligence*, pp. 1366–1373, 2015.

[16] E. Kindler, V. Rubin, and W. Schäfer. Incremental workflow mining based on document versioning information. In *Unifying the Software Process Spectrum*, pp. 287–301. Springer, 2006.

[17] E. Kindler, V. A. Rubin, and W. Schäfer. Incremental workflow mining for process flexibility. In *BPMDS*, 2006.

[18] V. Leno, A. Armas-Cervantes, M. Dumas, M. La Rosa, and F. M. Maggi. Discovering process maps from event streams. In *ICSSP*, pp. 86–95. ACM, 2018.

[19] A. Leontjeva, R. Conforti, C. D. Francescomarino, M. Dumas, and F. M. Maggi. Complex symbolic sequence encodings for predictive monitoring of business processes. In *BPM*, *LNCS* 9253, pp. 297–313. Springer, 2015.

[20] M. Li, M. Liu, L. Ding, E. A. Rundensteiner, and M. Mani. Event stream processing with out-of-order data arrival. In *ICDCSW'07*, pp. 67–67, 2007.

[21] A. Maaradji, M. Dumas, M. L. Rosa, and A. Ostovar. Fast and accurate business process drift detection. In *BPM*, *LNCS* 9253, pp. 406–422. Springer, 2015.

[22] M. Maisenbacher and M. Weidlich. Handling concept drift in predictive process monitoring. In *SCC*, pp. 1–8, 2017.

[23] C. Mutschler and M. Philippsen. Adaptive speculative processing of out-of-order event streams. *ACM Trans. Internet Technol.*, 14(1):4:1–4:24, 2014.

[24] J. E. Rumbaugh. A parallel asynchronous computer architecture for data flow programs. Technical report, MIT, 1975.

[25] A. Senderovich, A. Rogge-Solti, A. Gal, J. Mendling, and A. Mandelbaum. The ROAD from sensor data to process instances via interaction mining. In *CAiSE*, *LNCS* 9694, pp. 257–273. Springer, 2016.

[26] M. Stonebraker, U. Cetintemel, and S. Zdonik. The 8 requirements of real-time stream processing. *SIGMOD Record*, 34:42–47, 2005.

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

[28] S. J. van Zelst, A. Bolt, M. Hassani, B. F. van Dongen, and W. M. P. van der Aalst. Online conformance checking: relating event streams to process models using prefix-alignments. *Int. J. Data Sci. Anal.*, 8(3):269–284, 2019.

[29] S. K. L. M. vanden Broucke, J. Munoz-Gama, J. Carmona, B. Baesens, and J. Vanthienen. Event-based real-time decomposed conformance analysis. In OTM, *LNCS* 8841, pp. 345–363. Springer, 2014.

[30] M. Weidlich, H. Ziekow, J. Mendling, O. Günther, M. Weske, and N. Desai. Event-based monitoring of process execution violations. In BPM, *LNCS* 6896, pp. 182–198. Springer, 2011.

Classifying process deviations with weak supervision

Manal Laghmouch
Faculty of Business Economics
UHasselt - Hasselt University
Hasselt, Belgium
manal.laghmouch@uhasselt.be

Mieke Jans
Faculty of Business Economics
UHasselt - Hasselt University
Maastricht University
Belgium - The Netherlands
mieke.jans@uhasselt.be

Benoît Depaire
Faculty of Business Economics
UHasselt - Hasselt University
Hasselt, Belgium
benoit.depaire@uhasselt.be

***Abstract*—Although conformance checking is great at detecting process deviations, it still poses challenges that hinders adoption in auditing practice. A major challenge is that in real life a large number of deviating cases is often detected of which only a small amount are true anomalies and thus of real interest to auditors. The number of deviations are often too large to inspect one by one, which explains why auditing requires a sample-based approach. This paper contributes to the research on the practical feasibility of continuous auditing and studies the potential of weak supervision to classify deviations into anomalies and exceptions, allowing auditors to do a full-population analysis of the identified deviations. The Snorkel framework is applied which uses a set of imperfect domain expert rules to classify the set of deviations into anomalies and exceptions. A controlled and artificial experiment has been set up to explore the relation between the performance of this approach and the number and quality of domain expert rules. The results demonstrate the potential of this approach as a limited number of medium to high quality domain expert rules succeeds to classify deviations with acceptable accuracy.**

***Index Terms*—auditing, conformance checking, declare, process deviation, snorkel, weak supervision**

I. Introduction

Process mining is a family of process analysis techniques that discovers a process model from real process executions (process discovery), detects deviations between a normative model and the real process (conformance checking) and enhances the process (process enhancement) [1].

Process mining can be useful for auditing purposes to get a better understanding of the audited company's business environment, including the business processes. Mainly *conformance checking* has great potential in auditing [2] as it provides a tool to automatically discover deviations between the normative process model and the actual process [1]. However, various challenges remain which hinder full adoption in practice.

A major challenge relates to the nature of identified deviations. Deviations can be classified into two types, i.e. *exceptions* which are deviations from the normative model but acceptable because of specific conditions that hold [1] and *anomalies* which represent deviations that are an issue and require follow-up by the auditor [3], [4]. Theoretically, the auditor could check each deviating case to verify it can be classified as an exception or anomaly, but in practice this quickly becomes infeasible because of the large number of deviations identified by conformance checking [5]. Therefore, current conformance checking approaches do not entirely simplify the auditor's work, which prevents the realization of full-population auditing [4].

In this paper, we study the potential of a combination of conformance checking, weak supervision and domain expert rules as a step forward towards auditing the complete set of transactions of a specific business process, replacing up-front sampling methods [4]. The specific problem at hand is where the auditor is confronted with a large set of deviating cases, as identified by the conformance checking approach, which need to be classified as exceptions or anomalies. In the envisaged solution, the auditor only needs to provide a limited set of rules they would use to label deviating cases as exceptions or anomalies in combination with a weak-supervision approach to use this domain knowledge to label all deviations in the initial set. *Weak supervision* is a branch of machine learning that uses noisy, limited, or imprecise sources and while labels derived from such sources are also 'weak' (but inexpensive), weak supervision manages to construct powerful prediction models [6], [7].

We build upon the weak supervision framework Snorkel [8], to classify process deviations into anomalies and exceptions based on a limited set of expert rules. We analyze its potential in an experimental and artificial setting which replicates the context of a procure-to-pay process. For our approach, we opted for the Snorkel Framework as it allowed the injection of domain knowledge in a machine learning model by means of rules and we could extract the labeling model functionality from the larger framework. Auditor's expertise, encoded as *labeling functions*, is used to guide the classification process [8].

Our study shows how a realistic set of deviating cases can be classified into anomalies and exceptions without the need for manually evaluating each case individually or the need for a large set of labeled cases to train a classification model

[1]Because normative models are typically simplified and idealized representations of how a process should look like, such specific conditions are often not modeled along, which results deviations that are only exceptions rather than anomalies.

978-1-7281-9832-3/20/$31.00 ©2020 IEEE
DOI 10.1109/ICPM49681.2020.00023

(which also requires manual labeling work). The contribution of this paper is three-fold:

1) We demonstrate the potential of weak supervision, and the Snorkel framework in particular, to bring full-population based auditing closer to reality.
2) We demonstrate that Snorkel's labeling model has a higher performance, in terms of accuracy, than a majority voting regardless of the quality of the expert rules.
3) We indicate that six high to medium quality rules per label category suffice to identify 60% of the anomalous cases and reach an overall accuracy of 80%.

This paper is structured as follows. In Section II, we explain how the weak supervision system Snorkel can be used in auditing practice. Our research design is explicated in III. Section IV shows the quantitative results of our study. Section V discusses our findings and highlights the implications. In Section VI, we provide some related work. The paper is concluded in Section VII.

II. Snorkel for auditing

The machine learning system Snorkel can help to overcome the challenges related to conformance checking in auditing. Traditional supervised machine learning models need a large set of labeled training data to train predictive models. Snorkel does not. Instead, it combines weak supervision sources for training [8]. In this section, we explain how Snorkel can be used to classify process deviations in the context of auditing.

A. Theoretical background

A recent paradigm for weak supervision that does involve domain knowledge in machine learning models is *data programming*. Data programming combines the labels from many weak supervision sources, like heuristic rules, to label datasets programmatically without using any ground truth [9]. Snorkel is a system that implements the idea of data programming. It trains models without manually labeling any training data. The main principle behind Snorkel is that domain knowledge, encoded as *labeling functions*, is used for labeling purposes [8].

In this study, we implement Snorkel in the context of labeling cases as either anomalous (1) or exceptional (0). The labeling procedure needs deviating cases detected by conformance checking, in the form of an event log, as input. The event log contains a set of cases, which represent sequences of one execution of a process [1]. The log consisting of only deviating cases is unlabeled, meaning that the cases are not yet labeled as 0 or 1. The objective of applying Snorkel to an unlabeled set of deviating cases is to identify which cases are anomalous and which are exceptional. In what follows we explain how Snorkel works by walking through two steps: (1) Add a label to each case for each labeling function, and (2) Combine labeling function outputs in a generative model.

1) Add a label to each case for each labeling function: In the first step, auditors need to provide Snorkel with labeling functions. Labeling functions are rules that encode domain knowledge. They enable auditors to inject their knowledge into machine learning models. Therefore, it is important that the auditor has enough knowledge to explicitly determine rules that identify anomalous and exceptional cases. A more formal representation of a labeling function λ_i is given below.

$$\lambda_i \longmapsto \{-1, 0, 1\}$$

Each λ_i takes a deviating case and labels it as either exceptional (0), anomalous (1) or it can abstain (-1) from labeling. To illustrate the concept of abstaining, we define the following labeling function: "if a signature is missing, then the case is anomalous.". The example labeling function identifies whether a case is anomalous (1) or not. If the case is not anomalous, according to this labeling function, it is not necessarily exceptional (0). The abstain label (-1) enables modeling that type of behaviour.

One specific labeling function labels a subset of the data. More labeling functions can overlap (agree) or conflict (disagree) with each other. Notice that each labeling function labels each case in the given event log. The output label that an individual labeling function provides for a specific case, is stored in a $j \times i$ labeling matrix L, with j being the number of cases in the event log and i the number of labeling functions. The labeling matrix L is used as input for the next step.

In Algorithm 1, we provide the pseudo-code that illustrates how we implemented this step in Snorkel.

Algorithm 1

Given:
cases: set of deviating cases in event log

Let:
l(i) be a labeling function with index i, defined by an auditor
L(i,j) be a label outputted by labeling function l(i) for case j

```
for labeling function l(i) do
  for case j do
    if labeling function l(i) outputs anomaly then
      Store 1 in L(i,j)
      else if labeling function l(i) outputs exception then
      Store 0 in L(i,j)
      else Store -1 in L(i,j)
    end if
  end for
end for
```

2) Combine labeling function outputs in a generative model: In the previous step, each of the j case is labeled by each of the i labeling functions as either an anomaly (1), an exception (0) or unknown (-1), resulting in an $j \times i$ labeling matrix L. As these labeling functions are imperfect and often local rules, labels for a specific case can contradict and two random cases are often labeled by a partially differing set of labeling functions. The goal of a labeling model is to take the i labels for a specific case and map it to a final predicted label.

A first naive approach is to use a 'majority vote' labeling model, which predicts the label most often assigned by the labeling functions to the specific case. This approach can become particularly troublesome as labeling functions are

correlated as this increases the weight of some rules in the set. In other words, correlated labeling functions are together more influential when assigning the final label than uncorrelated labeling functions. Furthermore, this also ignores the fact that these labeling functions are of varying quality and some rules should not receive too much weight.

The Snorkel framework follows a different approach by estimating a generative model for the unlabeled data, in order to derive the appropriate weights for each labeling function. Intuitively, Snorkel models the true label for a case as a latent variable in a probabilistic model. It encodes a generative model using parameters which represent the probability that a rule either labels a case as an anomaly or exception rather as unknown and the probability that a rule makes a correct prediction. Next, these parameters are learned on the unlabeled data set by minimizing the negative log marginal likelihood. Ultimately, the generative model uses the accuracies and correlations of the labeling functions to combine the individual output labels into a single confidence-weighted label per case. For technical details on *data programming* and *Snorkel*, we refer to works of [8], [9].

In Algorithm 2, we provide the pseudo-code of this step.

Algorithm 2

Given:
j: case index
i: labeling function index
L: $i \times j$ label matrix, consisting of labeling function outputs

Let:
y(j) be the final output label for case j

for labeling function l(i) **do**
  for case j **do**
    Learn labeling function accuracies
    Learn labeling function correlations
    Reweigh labeling functions
    Calculate final label
    Add final label to y(j)
  end for
end for

B. Assumptions

With Snorkel, we aim to label the unlabeled event log by using labeling functions. Since labeling functions represent the auditor's domain knowledge, the auditor's role is essential in our study. Consequently, an important assumption in testing whether weak supervision systems can be used to classify deviations, is that an auditor is capable of defining labeling functions that identify anomalous and exceptional cases. Notice that this shifts the auditor's role from checking individual cases on anomalies or exceptions to providing high level rules that identify anomalous or exceptional cases.

C. Implementation

Snorkel requires Python 3.6 or a later version. We implemented Snorkel and the algorithms from Sections II-A in Python 3.6 for Linux. The codes are stored in a GitHub repository[2]

III. Experimental design

This section explicates the research design of our study. First, we explain our research goal and design. Next, we describe how data were artificially generated. Final, we provide some background on how we constructed labeling functions.

A. Research goal and design

We define a set of deviating cases that needs to be classified as anomalous or exceptional. More specifically, we generate an unlabeled set of deviating cases and a labeled validation set that is used to validate our results. Subsequently, we use heuristics, representing auditing knowledge, as sole input to label the unlabeled set of deviating cases.

In order to test how good the weak supervision system Snorkel performs at classifying deviating cases as anomalous or exceptional, we run a set of computational tests under controlled conditions. We want to measure whether following parameters have an impact on the model's performance:

- The quality of the labeling functions, expressed in terms of accuracy
- The number of labeling functions

Above parameters are both associated with labeling functions. To determine their quality, we first construct three sets of different quality levels of labeling functions: high ($> 80\%$), medium ($60\% - 80\%$) and low ($< 60\%$) accuracy labeling functions. Every set consists of at least 10 labeling functions. A labeling function is a proxy for the auditor's domain knowledge. Therefore, respectively, a high, medium and low accuracy labeling function represents that an auditor is very good, medium or not good at all at defining labeling functions that accurately express their knowledge.

We chose for an artificial setting because this research is still in an exploratory phase. It is expensive to collect data when it is unclear which relations we are looking for. Therefore, it is more logical to start with an artificial, but realistic, setting in which we can control the data. Building upon this, it is important to know what the data comprises, because there is a high risk that some patterns in the data point to correlations (rather than causal relations). This can lead to a wrong interpretation of the results. Furthermore, the artificial setting enables us to simulate auditors of different expert levels by defining different quality levels of rules. Nevertheless we call the artificial setting 'realistic', because we start from a specific and realistic setting: a procure-to-pay process.

B. Data

The set of deviating cases, including exceptions and anomalies, that is used in this study is artificially generated. This allows, amongst other things, for a set-up with a validation set. To this purpose, we start from three declarative process models. We created a *normative*, an *auditor* and a *real* process

[2]https://github.com/manallaghmouch/snorkelforauditing

model for a procure-to-pay process; the first being the most restrictive, and each of the latter two being a less restrictive version of the previous one. The models consist of eight activities: *create purchase request*, *approve purchase request*, *create purchase order*, *sign*, *receive goods or services*, *receive invoice* and *pay*.

The normative process model represents the process that a company desires to follow and holds the most restrictions. It contains 14 declarative constraints in our design. Deleting some of these constraints results in the auditor process model (7 constraints). This model contains the process paths that the auditor would accept, even if they deviate from the normative process model. Deleting again some declarative constraints from the auditor model results in the real process model. This is the least restrictive model of the three, and consists of the lowest number of declarative constraints (3 constraints). Behaviour that fits in this model, but not in the auditor model, are examples of deviating cases that are anomalies. The declarative constraints in the real model simulate the user constraints that are imposed by the information system's configuration. As such, the behaviour that is modeled in the real process model represents the as-is process.

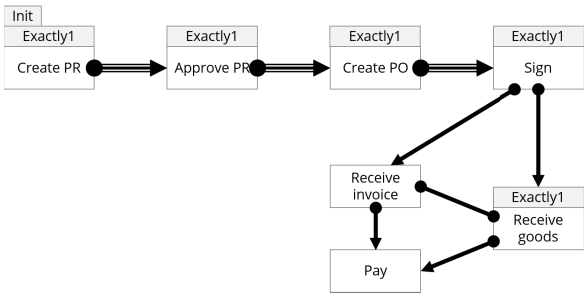


Fig. 1: The designed normative process model.

To get a better understanding of how the designed Declare models look like, we show the normative process model in Figure 1. An example of a declarative constraint in this model is a chain response between the activities *Create PR* and *Approve Pr*. This means that if *Create PR* occurs, then *Approve PR* directly follows. The auditor model is not visualised, but consists of seven constraints that are a subset of the normative process model. Following the same rationale, the real process model is a subset of the auditor model and consists of three constraints. For more information on the DECLARE template, we refer to the work of [10].

Recall that the declarative process models were designed with the sole purpose to artificially create an unlabeled event log and a validation set. We used the *real* process model to artificially generate two event logs using the *MinerFUL Log Generator* from [11]: one log of 2000 cases and another log of 1000 cases. Both logs contain cases of 3 to 10 events per case. The first event log represents the unlabeled event log, meaning that this set consists of cases that are not yet labeled as anomaly or exception. The second event log is used to obtain the validation set. Because the validation set is used to validate the outcomes of the labeling algorithm, it should consist of only labeled cases. We obtain the labels of the validation set cases by performing a conformance check in the following two steps:

1) The event log of 1000 cases is compared with the *normative* process model. The result is a binary classification into *deviation* and *no deviation*. The cases that contain at least one deviation are extracted as 'deviating cases'.
2) The deviating cases from step 1 are compared with the *auditor* process model. The result of this conformance check is a binary classification into *anomaly* and *exception*.

The cases that are labeled as 'anomaly' or 'exception' constitute the validation set. After the two-step conformance check, we obtain the following two datasets. (1) An unlabeled event log consisting of 2000 deviating cases, and (2) A balanced, validation set consisting of 1000 deviating cases (452 anomalies, 548 exceptions). Note that none of the 1000 artificially generated traces of the *real* process model fell within the constraints of the *normative* process model. As a consequence, all 1000 cases are deviating cases and are part of the validation set.

C. From rule sets to labeling functions

We define a *rule set* as a grouping of one or more rules that identify a concept. Because we are dealing with a binary classification problem, we need two key rule sets, respectively a set of rules that identify an anomaly (Rule Set Anomaly - RSA) and a set of rules that identify an exception (Rule Set Exception - RSE). In order to obtain the two key rule sets, we construct four additional rule sets: Rule Set Normative (RSN), Rule Set Auditor (RSAU), Rule Set Real (RSR) and Rule Set Deviations (RSD). RSN is the collection of declarative rules that constitute the normative process model. RSAU is the collection of declarative rules of the auditor process model. RSR is the collection of declarative rules of the real process model. RSD is the difference between the declarative rules in the normative process model and the declarative rules in the real process model. Mathematically, the rule sets are formulated as follows.

RSN = {x : x is declarative rule from normative model}
RSAU = {x : x is declarative rule from auditor model}
RSR = {x : x is declarative rule from real model}
RSD = RSN $\setminus$ RSR

RSA and RSE are then as follows.

RSA = RSD $\setminus$ RSAU
RSE = RSAU $\setminus$ RSN

IV. Results

In this section, we show the results of our experiments. First, we describe the effect of the quality of the labeling functions on the accuracy of Snorkel's labeling model, the LabelModel. Subsequently, we show the effect of the amount of labeling functions on the accuracy of the LabelModel.

A. Effect of the quality of labeling functions on performance

As described in the previous section, we constructed three sets of different quality levels of labeling functions: high, medium and low. Furthermore, we distinguished between anomaly and exception rules.

To test the effect of labeling function quality on the performance of the LabelModel, we limit the amount of labeling functions to a balanced set of five anomaly and five exception functions that were randomly chosen. How the labeling function sets are composed in terms of quality level, differs over the experimental runs.

We executed 66 experimental setups in total. We can divide this in six parts of 11 setups. The 11 setups were structured as follows. We fixed the amount of the anomaly (exception) rules to five at a constant level of quality, while gradually decreasing the quality of the exception (anomaly) rules. For example, we start with 5 high quality anomaly rules and 5 high quality exception rules. In the second setup, we still have 5 high quality anomaly rules, but 4 high quality exception rules and 1 medium quality exception rule. In the third setup, we again still have 5 anomaly rules, but 3 high quality exception rules and 2 medium quality exception rules. We keep on replacing higher level exception rules by lower level exception rules until we are left with 5 low quality exception rules.

Each setup was repeated 15 times to obtain an average accuracy. The mean accuracies of Snorkel's LabelModel and the naive approach of majority voting were calculated. Figure 2 visualizes our findings. Notice that the 'lift' in the Figures should be interpreted as the improvement the LabelModel offers relative to the naive approach of majority voting. To get an understanding of the behaviour of labeling function quality per category on the accuracy of the models, we kept one category fixed, while decreasing the quality of the labeling rules of the other category. For example, in (a), we show the effect of the quality of exception rules on the accuracy, keeping anomaly rules at a constant level of high quality.

Figure 2 shows that Snorkel's LabelModel performs significantly better in classifying deviating cases than the naive approach of majority voting ($t = 0.00, p < 0.01$). High- and medium-quality anomaly rules are robust to changes in quality of the exception rules (Figure 2 (a) and (b)). Furthermore, for the fixed medium- and low-quality anomaly rules, the LabelModel decreases abruptly at the point where the quality level of exception rules shifts from high quality to medium/low quality. The same reasoning holds for the fixed high- and medium-quality exception rules, where the quality level of the anomaly rules shifts from high quality to medium/low quality. This effect is mainly noticeable for the LabelModel. It suggests that the LabelModel loses most performance when the set of labeling functions consists of about 25% low-quality rules. However, adding even more low-quality rules to the model at the expense of high-quality rules does not have an additional decreasing effect on accuracy. All figures have about the same gradient, except for (c). This figure shows that majority voting may perform better than the LabelModel in some specific cases. A possible explanation for this behaviour in our setting is that the set of low quality exception rules contains rules with a zero accuracy, while the set of anomaly rules does not.

B. Effect of the number of labeling functions on performance

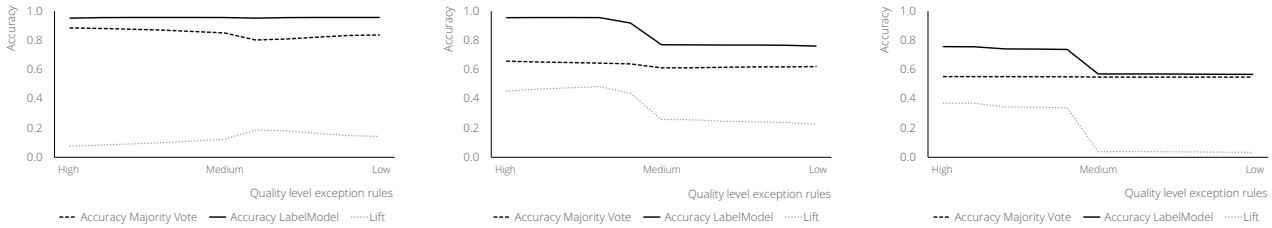
In order to test the effect of the number of labeling functions on the performance of the LabelModel and the majority voting approach, we increase the number of randomly chosen labeling functions gradually. We do this for an equal mix of high and medium quality labeling functions from the anomaly and exception categories. Since Snorkel requires a minimum of 3 labeling functions, our minimum is set to 4 (to keep a balanced set of anomaly and exception rules). For every setup, we increased the number of labeling functions per category by one.

First, we test the sensitivity of our model by showing how many anomalies the model classifies correctly with a balanced set of only high- and medium-quality anomaly rules as input. We focus on anomaly rules, because discovering anomalous cases is highly important from an auditing perspective. Second, we perform a similar analysis on a balanced combination of both anomaly and exception rules. In Figure 3, we graphically present our results.

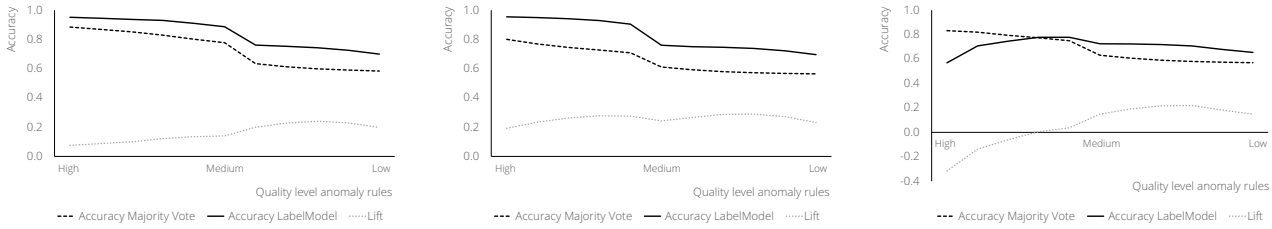
As both (a) and (b) show, the LabelModel is better at identifying anomalous cases than majority voting, regardless of the number of labeling functions and regardless the use of only anomaly rules or a combination of anomaly and exception rules (lift). When only using anomaly rules, the LabelModel correctly classifies 65% to 80% of the anomalies (a), when using both anomaly and exception rules, the LabelModel correctly classifies 50% to 75% of the anomalies (b). Furthermore, for both setups, is significantly better at correctly classifying anomalies than majority voting ($t = 0.00, p < 0.01$). The difference is more noticeable when both anomaly and exception rules are taken into account. However, the LabelModel labels more anomalous cases correctly when the model is provided with anomaly rules only. This might indicate that including additional exception rules, on top of the anomaly rules, in the model has a negative impact on correct classification.

Although from an auditing point of view, we focus on correctly classifying anomalous cases, we are still interested in predicting both anomalies and exceptions. Classifying both is important because at the end of the classification, a limited amount of cases might still be unlabeled. In the case of auditing, we would see these cases as 'potentially anomalous', to depict that the model was unsure about the label and therefore abstained from making a final decision on the label. To avoid the negative effect of having a balanced set of anomaly and exception rules on correctly classifying anomalies, we could provide the model with more anomaly rules than exception rules or assign more weight to the anomaly rules.

We also have to take into account that adding more rules to the model diminishes the added value when the model already consists of about 12 rules. With 12 rules consisting of a balanced set of high- and medium-quality rules, the LabelModel identified about 60% of the anomalous cases



(a) High-quality anomaly rules. (b) Medium-quality anomaly rules. (c) Low-quality anomaly rules.



(a) High-quality exception rules. (b) Medium-quality exception rules. (c) Low-quality exception rules.

Fig. 2: The effect of decreasing the quality of rules on accuracy.

correctly (twice as much than majority voting), the other 40% were not identified (i.e. the model abstained from labeling). The overall model accuracy at this point was 80%, meaning that many cases were correctly classified as anomalous or exceptional.

V. Discussion

This section points out the relevance and implications of our results in a broader perspective. We discuss the implications for research and practice and provide an overview of some research limitations.

A. Implications

Our work has implications for the research field of continuous auditing. Research in this area focuses on automation of tasks that highly require domain expertise. A general question here is to what degree full automation is practical in auditing practice. Recent studies suggest that automation could have a powerful effect on today's accounting practices [12]. However, it is less likely that auditing tasks will be fully automated. Rather, auditors will work together with machines to complete a task [13], [14]. The findings of our research implicate the same. Computational tools and techniques will only be valuable in auditing as long as using them significantly simplifies current auditing tasks, which is currently not the case. Conformance checking results in a large set of process deviations and forces an auditor to take samples [4], [5], [15].

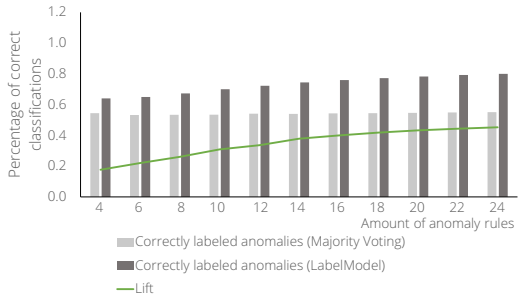
Our study shows that weak supervision can offer a solution for the problem of alarm floods in auditing, enabling full population testing. We demonstrate that the weak supervision system Snorkel smoothly integrates domain knowledge, in the form of rules, in a machine learning model. Our results suggest that the provided rules have to be of medium to high quality to obtain reasonable results. Furthermore, with only six high- to medium-quality labeling rules for both anomalies and exceptions, the Snorkel model reaches an acceptable accuracy.

B. Limitations

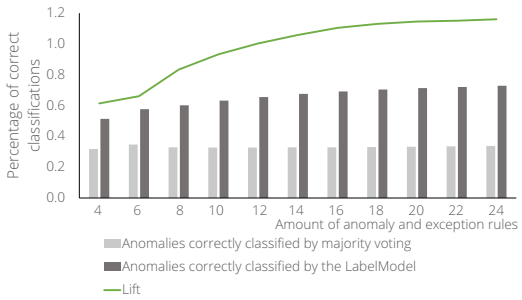
Our study has some limitations that are inherent to its artificial setup. First, the generated event logs are based on a procure-to-pay process. While we designed a realistic process model and obtained powerful results, future research has to investigate the potential effect of using different process models. A reasonable starting point for such research can be changing the complexity of the declarative model and explore its effect on model performance.

Second, although the declarative rules are a good proxy of an auditor's knowledge, it does not fully account for an actual auditor. Now we know weak supervision potentially holds a solution for continuous auditing, a follow-up fine-tuning of our research direction might be as follows. As a first step, we can design a semi-artificial experiment in which auditors have to verify rules from the auditor model. Subsequently, we let auditors design rules from scratch and use these as labeling functions.

Third, to label the generated set of deviating cases, we used Snorkel's generative model. While this model effectively labeled our dataset, a discriminative model could generalize beyond the provided heuristics without losing precision [8]. The obtained (labeled) event log can serve as input to train a



(a) Correctly classified anomalies, based on anomaly rules.



(b) Correctly classified anomalies, based on anomaly and exception rules.

Fig. 3: The effect of increasing the number of rules on correct classification of anomalies.

discriminative model on unseen data. The possibilities in the context of auditing have to be further investigated in future research.

VI. Related work

This section describes background literature that is relevant to our research. We start with discussing prior research on conformance checking in auditing. Thereafter, we provide some background on weak supervision and Snorkel.

A. Continuous auditing of business processes

The increased availability of digital data and information systems is changing the current way of auditing [16]–[18]. Continuous Auditing is the idea of automating audit procedures. Such techniques can assist auditors in objectively auditing a company [4], [16], [19], [20]. Continuous auditing enforces traditional auditing to change. Auditors who are familiar with the flow of transactions and related control activities become essential, as they can now analyze business processes based on digital transactions [21].

Many continuous auditing tools and techniques can be used to analyze business processes [22], [23]. In particular, process mining is a family of process analysis techniques that provides an objective view on the process. It uses an event log as input to initialize automatic analysis [1]. Conventionally, process mining is divided in three types: process discovery, conformance checking and process enhancement. With process discovery, a process model is discovered from real process executions. With conformance checking, a normative process model is compared with the real process to detect process deviations. With process enhancement, additional analysis like time and resource analysis are performed with the objective to enhance the actual process. Using solely an event log as input, process mining techniques can provide insightful information about a business process, like sequence, duration and interaction information [1], [24].

Mainly the type *conformance checking* is useful for more in-depth auditing analyses [2], [4]. Although conformance checking smoothly automates the identification of a large set of process deviations, using this in auditing practice bears some challenges. Since the normative process model is a simplified representation of how the process should look like, an alarm flood of process deviations is detected. This set consists of a relatively small set of deviations that are important from an auditing point of view (anomalies) and a relatively large set of technical deviations that do not matter from an auditing perspective (exceptions) [4]. Current conformance checking output does not simplify the auditor's work, with the consequence that sampling is still necessary. Some studies propose frameworks that could provide a solution for alarm floods in continuous auditing [4], [15], [25]. However, a practical integration of existing techniques is not yet tested. In this paper, we demonstrated and experimentally tested how a weak supervision system can be leveraged to classify deviating cases as anomalous or exceptional.

B. Weak Supervision

Machine learning based systems are getting increasingly more attention nowadays. With deep learning techniques, can effectively be used to predict the labels of unseen data instances. However, those techniques require large training sets of labeled examples to reach high predictive performance. Obtaining these large sets of labeled data is time consuming and costly [26]. Because of this, weak supervision is gaining more attention. It is a branch of machine learning in which cheaper sources of heuristic or noisy labels are used [27]. Some popular forms of weak supervision are distant supervision [28], [29], crowd-sourced labeling [30], [31], and heuristics for labeling data [32]. Although these weak supervision forms are inexpensive, they have a rather limited accuracy [27].

A recent paradigm for weak supervision that has been proposed in literature is *data programming*. Data programming combines the labels from many weak supervision sources, without using ground truth labels, to increase the accuracy and coverage of training data. Furthermore, probabilistic training labels that represent the lineage of individual labels, are

generated. Consequently, with only a generative mode, source accuracies and correlation structures are recovered without using labeled training data Snorkel is a system that implements data programming to train machine learning models without manually labeling any training data. For this purpose, it uses a set of labeling functions (i.e. rules) as input [8]. A labeling function encodes domain knowledge and other supervision sources programmatically. After defining a set of labeling functions, the labeling functions are combined in a machine learning model to build labeled training sets that can be used to train a discriminative model [8], [27]. Since domain knowledge is very crucial in auditing and Snorkel provides a way of injecting domain knowledge into machine learning models, we used the Snorkel system in this paper to demonstrate its potential to classify process deviations.

VII. Conclusion

In this paper, we show how weak supervision effectively labels deviating cases as *anomalous* or *exceptional*. We did this by implementing the weak supervision system Snorkel. We ran a set of computational experiments on a realistic, but artificially generated, event log of a procure-to-pay process, along with a normative and auditor model, representing the desired business process on the one hand and the acceptable process on the other hand. The results of this study suggest that Snorkel's labeling model performs better than the naive approach of majority voting. With each six high- to medium-quality labeling functions for both anomalies and exceptions, the Snorkel model reaches a relatively high accuracy of 80% and identifies at least 60% of the anomalous cases.

References

[1] W. Van der Aalst, *Process mining: Data Science in Action*, 2016.

[2] M. Jans, M. Alles, and M. Vasarhelyi, "The case for process mining in auditing: Sources of value added and areas of application," *International Journal of Accounting Information Systems*, vol. 14, no. 1, pp. 1–20, 2013.

[3] B. Depaire, J. Swinnen, M. Jans, and K. Vanhoof, "A process deviation analysis framework," in *International Conference on Business Process Management*. Springer, 2012, pp. 701–706.

[4] M. Jans and M. Hosseinpour, "How active learning and process mining can act as Continuous Auditing catalyst," *International Journal of Accounting Information Systems*, vol. 32, pp. 44–58, Mar. 2019.

[5] M. G. Alles, A. Kogan, and M. A. Vasarhelyi, "Putting continuous auditing theory into practice: Lessons from two pilot implementations," *Journal of Information Systems*, vol. 22, no. 2, pp. 195–214, 2008.

[6] Z.-H. Zhou, "A brief introduction to weakly supervised learning," *National Science Review*, vol. 5, no. 1, pp. 44–53, 2018.

[7] A. Ratner, S. Bach, P. Varma, and C. Ré, "Weak supervision: the new programming paradigm for machine learning," *Hazy Research. Available via https://dawn. cs. stanford. edu//2017/07/16/weak-supervision/. Accessed*, pp. 05–09, 2019.

[8] A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré, "Snorkel: rapid training data creation with weak supervision," *The VLDB Journal*, vol. 29, no. 2, pp. 709–730, 2020.

[9] A. J. Ratner, C. M. De Sa, S. Wu, D. Selsam, and C. Ré, "Data programming: Creating large training sets, quickly," in *Advances in neural information processing systems*, 2016, pp. 3567–3575.

[10] M. Pesic, H. Schonenberg, and W. M. Van der Aalst, "Declare: Full support for loosely-structured processes," in *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*. IEEE, 2007, pp. 287–287.

[11] C. Di Ciccio, M. L. Bernardi, M. Cimitile, and F. M. Maggi, "Generating event logs through the simulation of declare models," in *Workshop on Enterprise and Organizational Modeling and Simulation*. Springer, 2015, pp. 20–36.

[12] C. B. Frey and M. A. Osborne, "The future of employment: How susceptible are jobs to computerisation?" *Technological forecasting and social change*, vol. 114, pp. 254–280, 2017.

[13] J. Mendling, G. Decker, R. Hull, H. A. Reijers, and I. Weber, "How do machine learning, robotic process automation, and blockchains affect the human factor in business process management?" *Communications of the Association for Information Systems*, vol. 43, no. 1, p. 19, 2018.

[14] N. Berente, S. Seidel, and H. Safadi, "Research commentary—data-driven computationally intensive theory development," *Information Systems Research*, vol. 30, no. 1, pp. 50–64, 2019.

[15] P. Li, D. Y. Chan, and A. Kogan, "Exception prioritization in the continuous auditing environment: A framework and experimental evaluation," *Journal of Information Systems*, vol. 30, no. 2, pp. 135–157, 2016.

[16] D. Y. Chan and M. A. Vasarhelyi, "Innovation and practice of continuous auditing," *International Journal of Accounting Information Systems*, vol. 12, no. 2, pp. 152–160, 2011.

[17] H. Chen, R. H. Chiang, and V. C. Storey, "Business intelligence and analytics: From big data to big impact," *MIS quarterly*, pp. 1165–1188, 2012.

[18] M. Jans, M. Alles, and M. Vasarhelyi, "The case for process mining in auditing: Sources of value added and areas of application," *International Journal of Accounting Information Systems*, vol. 14, no. 1, pp. 1–20, 2013.

[19] S. Groomer and U. Murthy, "Continuous auditing of database accounting systems using embedded audit modules," *Journal of Information Systems*, vol. 3, no. 1, pp. 53–69, 1989.

[20] M. A. Vasarhelyi, M. G. Alles, and A. Kogan, "Principles of analytic monitoring for continuous assurance," *Journal of emerging technologies in accounting*, vol. 1, no. 1, pp. 1–21, 2004.

[21] M. Werner, "Financial process mining-accounting data structure dependent control flow inference," *International Journal of Accounting Information Systems*, vol. 25, pp. 57–80, 2017.

[22] A. Datta, "Automating the discovery of as-is business process models: Probabilistic and algorithmic approaches," *Information Systems Research*, vol. 9, no. 3, pp. 275–301, 1998.

[23] S. X. Sun, J. L. Zhao, J. F. Nunamaker, and O. R. L. Sheng, "Formulating the data-flow perspective for business process management," *Information Systems Research*, vol. 17, no. 4, pp. 374–391, 2006.

[24] R. J. C. Bose and W. van der Aalst, "Trace alignment in process mining: opportunities for process diagnostics," in *International Conference on Business Process Management*. Springer, 2010, pp. 227–242.

[25] J. L. Perols and U. S. Murthy, "Information fusion in continuous assurance," *Journal of Information Systems*, vol. 26, no. 2, pp. 35–52, 2012.

[26] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, "Revisiting unreasonable effectiveness of data in deep learning era," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 843–852.

[27] A. J. Ratner, S. H. Bach, H. R. Ehrenberg, and C. Ré, "Snorkel: Fast training set generation for information extraction," in *Proceedings of the 2017 ACM international conference on management of data*, 2017, pp. 1683–1686.

[28] A. Madaan, A. Mittal, G. Ramakrishnan, S. Sarawagi *et al.*, "Numerical relation extraction with minimal supervision," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[29] A. Smirnova and P. Cudré-Mauroux, "Relation extraction using distant supervision: A survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 5, pp. 1–35, 2018.

[30] G. Xu, W. Ding, J. Tang, S. Yang, G. Y. Huang, and Z. Liu, "Learning effective embeddings from crowdsourced labels: An educational case study," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 1922–1927.

[31] R. Lotfian and C. Busso, "Curriculum learning for speech emotion recognition from crowdsourced labels," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 27, no. 4, pp. 815–826, 2019.

[32] C. De Sa, A. Ratner, C. Ré, J. Shin, F. Wang, S. Wu, and C. Zhang, "Deepdive: Declarative knowledge base construction," *ACM SIGMOD Record*, vol. 45, no. 1, pp. 60–67, 2016.

An Entropic Relevance Measure for Stochastic Conformance Checking in Process Mining

Artem Polyvyanyy
The University of Melbourne
Email: artem.polyvyanyy@unimelb.edu.au

Alistair Moffat
The University of Melbourne
Email: ammoffat@unimelb.edu.au

Luciano García-Bañuelos
Tecnológico de Monterrey
Email: luciano.garcia@tec.mx

***Abstract*—Given an event log as a collection of recorded real-world process traces, process mining aims to automatically construct a process model that is both simple and provides a useful explanation of the traces. Conformance checking techniques are then employed to characterize and quantify commonalities and discrepancies between the log's traces and the candidate models. Recent approaches to conformance checking acknowledge that the elements being compared are inherently stochastic – for example, some traces occur frequently and others infrequently – and seek to incorporate this knowledge in their analyses.**

Here we present an *entropic relevance* measure for stochastic conformance checking, computed as the average number of bits required to compress each of the log's traces, based on the structure and information about relative likelihoods provided by the model. The measure penalizes traces from the event log not captured by the model and traces described by the model but absent in the event log, thus addressing both precision and recall quality criteria at the same time. We further show that entropic relevance is computable in time linear in the size of the log, and provide evaluation outcomes that demonstrate the feasibility of using the new approach in industrial settings.

I. Introduction

Process mining studies tools, methods, and techniques for improving real-world processes based on event data generated by historical process executions [1]. The core problem in process mining is that of automatically discovering a process model from a given *event log*, where an event log is a collection of traces, each capturing a sequence of observed process events. Such discovered models should faithfully encode the process behavior captured in the log and, hence, satisfy a range of criteria. Specifically: (1) a discovered model should describe as many as possible of the traces recorded in the log (good *recall*, or *fitness*); (2) should allow as few traces as possible that are not present in the log (good *precision*); and (3) should be as "*simple*" as is consistent with the other two goals (good *simplicity*). In tension with these three is a further objective: (4) the model should allow traces that may stem from the same process but are not present in the sample (good *generalization*). Conformance checking is a subarea of process mining that addresses the problem of measuring and characterizing the four quality criteria when using a discovered process model to explain the corresponding event log.

Classical process mining techniques often consider frequencies of traces in the logs and/or frequencies of events in the traces. However, the implications of such considerations usually stay hidden from the consumers of the artifacts that are produced. For instance, most existing discovery algorithms strive to construct process models that fit, i.e., can replay, frequent traces from the input event logs. However, the vast majority of the discovery techniques construct non-deterministic models in which routing decisions are equiprobable, significantly limiting the ability of the models to explain the behaviors of the true processes sampled via the event logs. Indeed, if traces in a log suggest that failure was observed in nine out of ten traces, a model that says that (only) every second trace is erroneous is of reduced utility, and potentially harmful to subsequent decision-making activities. By accumulating event data over a long time, the log approaches the true event and trace frequencies. If this information about the true process were able to be reflected in the discovered model, the model would generate better process simulations, and hence better predictions of future processes.

We refer to process mining endeavors that process and produce artifacts with explicit information on the likelihood of process events and traces collectively as *stochastic*, or *statistical*, *process mining*. For example, a discovery algorithm could construct a model with annotations of relative likelihoods of making routing decisions such that a large collection of traces induced by the model (by following the encoded stochastic decisions) would result in a probability distribution over traces that closely matches the distribution of log traces. We call such a model a *stochastic process model*. Before designing algorithms for discovering stochastic process models from event logs, we seek to understand which stochastic models can be considered to be "good" explanations of a given log.

Hence this paper, in which we present a novel technique for stochastic conformance checking called *entropic relevance*, or *relevance*. Given a log, the entropic relevance of a stochastic process model is the average number of bits used to compress [2] a trace from the log using the relative likelihoods induced by the model. The fewer bits are used, the better the model "explains" the event log, i.e., the closer the relative likelihoods of traces derived from the model to those obtained from the event log. Log traces that do not fit the model are penalized by encoding them using a more expensive background model. Entropic relevance also penalizes traces that fit the model but are not present in the log, as these reduce the likelihoods of the log traces that fit the model, and increase the length (in terms of bits) of their compressed forms. Hence, an entropic relevance measurement reflects a compromise between the

978-1-7281-9832-3/20/$31.00 ©2020 IEEE
DOI 10.1109/ICPM49681.2020.00024

precision and recall quality criteria. Such duality is indeed expected in the case of stochastic conformance checking. Finally, given a model and log, relevance is computable in time linear in the size of the log, making it – as we demonstrate below – useful when evaluating the quality of the process discovery algorithms commonly used in the industry.

The remainder of the paper proceeds as follows. The next section presents an overview of the compression methodology we employ, and describes how an event log can be encoded via a stochastic process model. Section III discusses formal stochastic models used in process mining. Based on these models, Section IV presents the notion of entropic relevance. Section V discusses the results of our evaluation of the entropic relevance measure, while Section VI summarizes related work.

II. Motivation and Overview

We employ a minimum description length compression-based framework to measure the quality of process models. Two key observations make this possible: that a good process model is one which accurately describes an observed set of traces, taken as a sample from an underlying universe of traces; and that the "describes" operation can be precisely quantified by assessing the cost of compressing the set of traces relative to the stochastic language expressed by the model. A signal benefit of this *entropic relevance* approach is that not only is the model *structure* an influence on its measured usefulness, but also the *probability* of each individual trace having emerged from the model. A relationship fundamental to information theory is critical to understanding the new approach: if a symbol e of probability $p(e)$ occurs, then the information conveyed by that occurrence is $-\log_2 p(e)$ bits, and hence that is also the minimum number of bits required to describe any instance of e. For example, if e has probability $p(e) = 0.8$, then each e that occurs in a stream of such symbols has a cost of 0.3219 bits attributable to it. Practical compression systems operate very close to these entropy-based limits, see, for example, Moffat and Turpin [3, Chapter 5]; and the information-theoretic relationship between probabilities and bits is an achievable one.

Fig. 1 provides an overview of our proposal. In the figure it is supposed that an event log has been provided, sampled from an underlying "true" (but unknown) process; and that two process models are being considered as alternative explanations for that set of observations. If each of the two models assigns a calculable probability to every possible sequence of process states that might occur, then computing $\sum_t -\log_2 p(t \mid M)$ values, where the summation is over the traces t in the log and $p(t \mid M)$ is the probability assigned to trace t by model M, results in a value that encapsulates the information content of the whole log, given M. The smaller that value, the shorter the compressed output would be were it to be generated, and the better the model matches the log. (That is, while Fig. 1 suggests that actual compression occurs, it is the *size* of the output that is of interest, and not the actual bits that arise.)

Fig. 2 provides more details of the compression process, considering each trace in the log. The stochastic process model

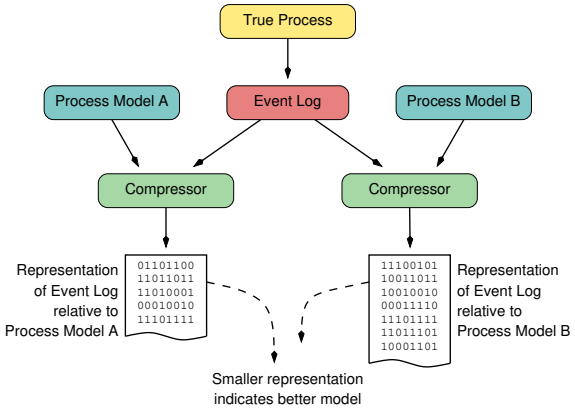


Fig. 1: To measure the entropic relevance of a process model to a collection of traces, the model's structure and probabilities are used to compute the cost in bits of losslessly representing the traces relative to the model model. Better models lead to a shorter compressed forms.

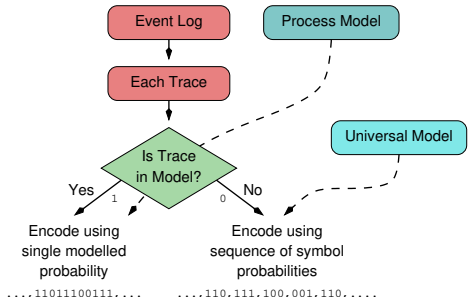


Fig. 2: The two possible options when encoding a set of traces with respect to a probabilistic model: each trace either has a non-zero probability in the model, which can be used to derive a bitstring; or it is spelled-out as symbol-by-symbol codes via a universal model.

(defined in detail in Section III) assigns calculable non-zero probabilities to a finite or infinite subset of the universe of possible traces, rather than to every possible member of that universe; and hence assigns a probability of zero to each of the infinite number of possible state sequences in the complement subset. If some particular trace in the log fits the model, it can be coded as single entity, using its corresponding end-to-end probability in the model. On the other hand, the traces with a probability of zero according to the model must be "spelled out" on a symbol-by-symbol basis, using a background (or *universal*) model in which every possible state always has a non-zero probability, and hence in which every possible sequence of states can always be coded. To choose between these two cases, the output associated with every trace is prefixed by a code – a (biased) 0 or 1 bit – that indicates which option applies. Given such an encoding and the stochastic and background models, one can always reconstruct the original log by applying the reverse procedure.

Fig. 3 steps back from the detail and provides a high-level view of the proposed mechanism. There are four components that collectively sum to the compressed size, and that vary in different ways as the process model changes. At the left

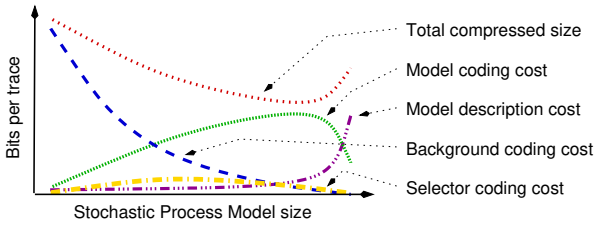


Fig. 3: Schematic showing the total compressed size of a collection of traces relative to a model as the sum of four components: the cost of describing the model and its parameters; the cost of entropy coding the traces that fit the model; the cost of entropy coding the traces that do not fit the model, using a catch-all background technique; and the cost of selecting, for each trace, which approach is used to code it.

end of the figure, if the process model is small and easily described, it likely fits only a small fraction of the log's traces. The majority of traces, the ones that do not fit the model, are coded using the more expensive background model, and it dominates to total cost. As the process model becomes larger and more sophisticated, it fits a greater fraction of the traces, and the balance shifts from the background model to the more economical stochastic process model. The total compressed size decreases as this transition takes place. Throughout this normal operating range the contributions of the other two factors – the binary per-trace selector flag, and the description of the process model – are typically very small overheads.

In the limit, at the right of Fig. 3, the process model becomes large and is over-fitted to the traces in the particular log. The cost of using an over-fitted model is low, since each pathway through it is unique; the cost of the background model is also low, since no sequences need to be processed via it; and there is no cost involved in selecting between the stochastic model and the background model. However the total compressed size will have increased, because of the complexity and detail required in the description of the process model.

In terms of Fig. 3, we define (see Section IV for full details) entropic relevance to be the sum of the selector coding cost, the background coding cost, and the model coding cost. It is useful to retain the model size (measured in some conventional manner) as a second dimension, shown as the horizontal axis in the plot. Furthermore, since process models are discrete objects (rather than a continuous phenomena) the separation between entropic relevance and model size allows definition of a Pareto frontier, i.e., the set of models that are either smaller in size, or superior in terms of entropic relevance, to other possible models.

Entropic relevance (again, in anticipation of Section IV) is measured in "bits per trace", with small values being preferable to large ones. The numeric range is open-ended, and it is neither desirable nor possible to normalize the measurement in any way to obtain a "0 to 1" range. Instead, it has meaningful units that clearly indicate the complexity of the process that is being represented by the model. With that understanding established, the process mining desiderata listed at the beginning of Section I can be considered: (1) traces not covered by the model must be coded using the background predictions, increasing the net bit cost; (2) processes permitted by the process model but not present in the log cause the imputed probabilities of traces that do occur to decrease, again increasing the net bit cost; and (3) simple models have smaller model description costs, decreasing the net bit cost. Objective (4) can also be accounted for, by noting that the background model is always available, so hitherto unseen traces can be accommodated, albeit with increased net bit costs.

III. Models of Stochastic Languages

This section introduces the notion of a stochastic language and several models, theoretical and those used in practice, that aim at encoding stochastic languages. The notion and models are used in the subsequent formal discussions and explanations of the conducted empirical evaluations.

A. Stochastic Languages

A *language* is a, possibly infinite, collection of finite sequences of *symbols*. These sequences are often referred to as *words*. In this work, we use words to encode observed processes. Hence, we refer to words as (process) *traces* composed of *actions* rather than symbols. Let Λ be a universe of *actions*. Then, Λ^* is the set of all traces over Λ. By ϵ, $\epsilon \in \Lambda^*$, we denote the empty trace. For example, set $X = \{\epsilon, \mathtt{a}, \mathtt{ab}, \mathtt{abc}, \mathtt{abcd}, \mathtt{abcdd}, \mathtt{abcde}\}$ defines a language of seven traces; we write $\mathtt{abc}$ to denote sequence $\langle \mathtt{a}, \mathtt{b}, \mathtt{c} \rangle$ when the context is clear.

A *stochastic language* is an assignment of probabilities to traces so that the assigned probabilities sum up to one, i.e., a stochastic language is a probability density function over traces. For example, a stochastic language might be used to encode the relative likelihoods of observing words in a book, or encountering traces in an event log of a software system. A stochastic language is defined as follows.

Definition III.1 (Stochastic language): A *stochastic language* L is a function $L : \Lambda^* \to [0, 1]$ for which it holds that:

$$\sum_{\sigma \in \Lambda^*} L(\sigma) = 1.0.$$

⌟

For example, $L_1 = \{(\epsilon, 0.5), (\mathtt{a}, 0.25), (\mathtt{ab}, 0.125), (\mathtt{abc}, 1/16), (\mathtt{abcd}, 1/32), (\mathtt{abcdd}, 1/64), (\mathtt{abcde}, 1/64)\} \cup \bigcup_{t \in \Lambda^* \setminus X} \{(t, 0.0)\}$, where set X is specified above. Given a trace $t \in \Lambda^*$ and a stochastic language L, $L(t)$ specifies the relative likelihood of a randomly drawn trace to be equal to t.

By $\hat{L}$, we denote the set of all traces possible according to L, i.e., $\hat{L} := \{t \in \Lambda^* \mid L(t) > 0.0\}$. We say that L is *finite* if and only if $\hat{L}$ is finite; otherwise L is infinite. It holds that $\hat{L}_1 = X$, i.e., the traces in X are all the possible traces according to L_1, and thus L_1 is finite.

B. Stochastic Deterministic Finite Automata

A stochastic deterministic finite automaton (SDFA) can be used to encode a stochastic language; here we adopt the definition of an SDFA from Carrasco [4].

Definition III.2 (Stochastic deterministic finite automaton): A *stochastic deterministic finite automaton* (SDFA) is a tuple $(S, \Delta, \delta, p, s_0)$, where S is a finite set of *states*, $\Delta \subseteq \Lambda$ is a set of *actions*, $\delta : S \times \Delta \to S$ is a *transition function*, $p : S \times \Delta \to [0, 1]$ is a *transition probability function*, $s_0 \in S$ is the *initial state*, and for each state $s \in S$ it holds that $\sum_{\lambda \in \Delta} p(s, \lambda) \leq 1.0$. ⌟

By $\mathcal{A}$, we denote the universe of SDFAs.

Fig. 4 shows an SDFA using graphical notation. In this notation, the states and transition function are visualized as circles and arcs, respectively. For instance, the SDFA shown in Fig. 4 has seven states $s_0 \ldots s_6$, and its transition function is defined by $\{(s_0, \mathtt{a}, s_1), (s_1, \mathtt{b}, s_2), (s_2, \mathtt{c}, s_3), (s_3, \mathtt{d}, s_4), (s_4, \mathtt{d}, s_5), (s_4, \mathtt{e}, s_6)\}$. Arcs are labeled by actions and transition probabilities. Hence, the arc from state s_0 to state s_1 with label "$\mathtt{a}(1/2)$" specifies that $(s_0, \mathtt{a}, s_1) \in \delta$ and $(s_0, \mathtt{a}, 0.5) \in p$. Consequently, the transition probability function is defined by $\{(s_0, \mathtt{a}, 0.5), (s_1, \mathtt{b}, 0.5), (s_2, \mathtt{c}, 0.5), (s_3, \mathtt{d}, 0.5), (s_4, \mathtt{d}, 0.25), (s_4, \mathtt{e}, 0.25)\}$. State s_0 is the initial state and, hence, is denoted by an arrow leading to it.

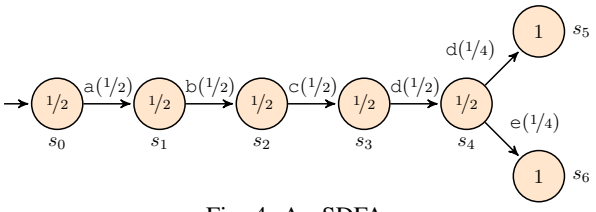


Fig. 4: An SDFA.

Given two traces $t_1, t_2 \in \Lambda^*$, by $t_1 \circ t_2$, we denote their concatenation, i.e., the trace obtained by joining t_1 and t_2 end-to-end. For example, it holds that $\mathtt{tr} \circ \mathtt{ace} = \mathtt{trace}$.

An SDFA $A = (S, \Delta, \delta, p, s_0)$ encodes stochastic language L_A defined using recursive function $\pi_A : S \times \Lambda^* \to [0, 1]$, i.e., $L_A(t) := \pi_A(s_0, t)$, $t \in \Lambda^*$, where:

$$\pi_A(s, \epsilon) := 1.0 - \sum_{\lambda \in \Delta} p(s, \lambda), \text{and}$$
$$\pi_A(s, \lambda \circ t') := p(s, \lambda)\, \pi_A(\delta(s, \lambda), t'), \lambda \in \Lambda, t = \lambda \circ t'.$$

Note that $\pi_A(s, \epsilon)$ denotes the probability of terminating a trace in state s of A. Such probabilities are shown diagrammatically as labels inside of the corresponding states. For example, for SFDA A from Fig. 4, it holds that $\pi_A(s_i, \epsilon) = 0.5$, $i \in [0\,..\,4]$, and $\pi_A(s_j, \epsilon) = 1.0$, $j = 5$ or $j = 6$.

If L is a stochastic language encoded by some SDFA, we say that L is a *regular stochastic language*. The SDFA in Fig. 4 encodes stochastic language L_1 discussed in Section III-A and, thus, L_1 is regular.

C. Event Logs

An *event log* is a finite collection of events that relate to a process and are distinguished by their attributes and attribute values. Usually, events in an event log encode information about actions executed by software systems that support a business process of an organization. In general, an event can have arbitrary attributes, but three attributes are common in process mining. These are the *case identifier*, *timestamp*, and the *action identifier* attribute. The value of the case identifier attribute of an event relates this event to a case, or instance, of the process; i.e., all events with the same case identifier stem from the same instance of the business process. The time of occurrence of an event is stored in its timestamp attribute. Finally, the action identifier attribute is used to store information about an action that induced the event.

In this work, we are neither interested in the exact times of event occurrences (but only in their orderings) nor in the distinctions between events and actions. Thus, we encode all the events with the same case identifier as a trace of corresponding actions (obtained via the action identifier attribute) arranged in the ascending order of the event timestamps. Finally, as there can be several case identifiers that induce the same trace (indeed, several business processes can induce the same sequence of actions with different timestamps), for our needs, it is convenient to represent an event log as a multiset of traces.

Definition III.3 (Event log): An *event log*, or *log*, is a finite multiset of traces. ⌟

By $\mathcal{E}$, we denote the universe of logs. For example, $E_1 = [\epsilon^{32}, \mathtt{a}^{16}, \mathtt{ab}^{8}, \mathtt{abc}^{4}, \mathtt{abcd}^{2}, \mathtt{abcdd}^{1}, \mathtt{abcde}^{1}]$ and $E_2 = [\epsilon^{250}, \mathtt{ab}^{250}, \mathtt{abc}^{250}, \mathtt{abcd}^{50}, \mathtt{abce}^{50}, \mathtt{abcde}^{50}, \mathtt{abced}^{50}, \mathtt{abcdde}^{50}]$ are two logs, i.e., $E_1, E_2 \in \mathcal{E}$. Trace $\mathtt{abc}$ occurs in E_1 four times, while in E_2 it is recorded 250 times.

An event log is inherently stochastic. By accumulating a large number of traces, and perhaps over an extended period of time, an event log aims to approach their true underlying probability distribution. Let X be a random variable and let O be a multiset of observations. By $P(X = x \mid O)$, or $P(x \mid O)$ when the context is clear, we denote the estimate based on O of the probability of observing X to be equal to element x. Given an event log $E \in \mathcal{E}$, we define the *stochastic language L of E* by assigning probability $L(t) := P(t \mid E)$ to each trace $t \in \Lambda^*$. In this work, we use the maximum likelihood estimation, i.e., $P(t \mid E) := {}^{m_E(t)}/_{|E|}$, where $m_E(t)$ denotes the multiplicity of element t in multiset E. Therefore, L_1 from Section III-A is the stochastic language of event log E_1 from above. Note that the stochastic language L_2 of event log E_2 is given my the function with these non-zero values $\{(\epsilon, 0.25), (\mathtt{ab}, 0.25), (\mathtt{abc}, 0.25), (\mathtt{abcd}, 0.05), (\mathtt{abce}, 0.05), (\mathtt{abcde}, 0.05), (\mathtt{abced}, 0.05), (\mathtt{abcdde}, 0.05)\}$.

D. Frequency Directed Action Graphs

A common approach for representing event logs for consumption and decision making by practitioners is by encoding them into Directly-Follows Graphs (DFGs) [1]. A DFG of an event log E is a digraph in which vertices are actions encountered in the traces of E and edges encode the directly-follows relation over the actions, i.e., the DFG contains an edge directed from action $\mathtt{a}$ to action $\mathtt{b}$ iff E contains a trace $t_1 \circ \mathtt{ab} \circ t_2$ where $t_1, t_2 \in \Lambda^*$ [5].

As DFGs of industrial event logs are immense, they are often post-processed by filtering out vertices and edges that

correspond, respectively, to infrequent actions and pairs of subsequent actions in the log. The vertices and edges of the filtered graphs are then annotated with numbers that reflect the frequencies of observing the corresponding concepts in the log. The frequencies aim to reflect the stochastic nature of the processes encoded in the corresponding log to inform the decision-making practices. To describe such filtered graphs mathematically, we introduce the notion of a *frequency directed action graph*.

Definition III.4 (Frequency directed action graph): A *frequency directed action graph* (FDAG) is a tuple $(\Phi, \Psi, \phi, \psi, i, o)$, where $\Phi \subseteq \Lambda$ is a set of *actions*, $\Psi \subseteq ((\Phi \times \Phi) \cup (\{i\} \times \Phi) \cup (\Phi \times \{o\}))$ is a *directly-follows relation*, $\phi : \Phi \cup \{i,o\} \to \mathbb{N}_0$ is an *action frequency function*, $\psi : \Psi \to \mathbb{N}_0$ is an *arc frequency function*, and $i \notin \Lambda$ and $o \notin \Lambda$ are the input and the output of the graph, respectively. ⌟

By $\mathcal{G}$, we denote the universe of FDAGs.

Fig. 5 shows an example FDAG. In the figure, boxes with rounded corners represent actions, whereas arcs encode the directly-follows relation. The input node and the output node are denoted by i and o, respectively. The input node has no incoming arcs, while the output node has no outgoing arcs. Finally, the action and arc frequencies assigned by the corresponding frequency functions are encrypted next to the respective actions and arcs.

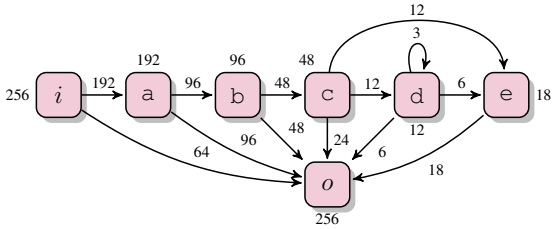


Fig. 5: An FDAG.

Van der Aalst [1] observes that practitioners can interpret FDAGs in different ways. Because of the filtering step, it is possible to associate a given FDAG with different collections of traces. He then discusses several pitfalls this phenomenon can lead to in practice. We agree with those observations, and, for our purpose, fix one such possible interpretation. To avoid ambiguities, next, we define our interpretation rigorously as a mapping from a given FDAG to the corresponding SDFA.

Definition III.5 (SDFA of FDAG): Let $G := (\Phi, \Psi, \phi, \psi, i, o)$ be an FDAG. Then, $(S, \Delta, \delta, p, s_0)$, where $S = \Phi \cup \{i\}$, $\Delta = \Phi$, $\delta = \{(s,t,t) \in (\{i\} \cup \Phi) \times \Phi \times \Phi \,|\, (s,t) \in \Psi\}$, $p = \{(s,t,x) \in (\{i\} \cup \Phi) \times \Phi \times [0,1] \,|\, (s,t) \in \Psi \wedge x = \psi(s,t) / \sum_{(s,u) \in \Psi} \psi(s,u)\}$, and $s_o = i$, is the SDFA of G, denoted by $\mathit{SDFA}(G)$. ⌟

Thus, if $A = \mathit{SDFA}(G)$, then, according to our interpretation, G encodes the possible traces of L_A, i.e., traces $\hat{L}_A$, and the relative frequency of each trace $t \in \hat{L}_A$ is given by $L_A(t)$.

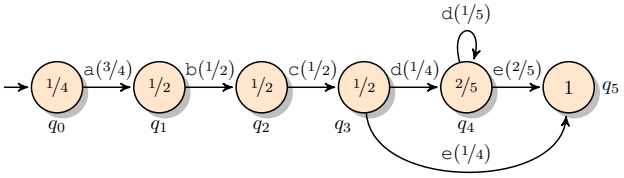


Fig. 6: An SDFA.

For example, Fig. 6 shows the SDFA of the FDAG from Fig. 5 that encodes infinite stochastic language L_2 which, among others, assigns these non-zero values to traces $\{(\epsilon, 1/4)$, $(\texttt{a}, 3/8)$, $(\texttt{ab}, 3/16)$, $(\texttt{abc}, 3/32)$, $(\texttt{abcd}, 3/160)$, $(\texttt{abce}, 3/64)$, $(\texttt{abcdd}, 3/800)$, $(\texttt{abcde}, 3/160)$, $(\texttt{abcddd}, 3/4000)$, $(\texttt{abcdde}, 3/800)\}$; all the other possible according to L_2 traces have the cumulative relative frequency of $9/8000$.

IV. Entropic Relevance

This section gives a precise definition of *entropic relevance*. Given an event log $E \in \mathcal{E}$ and an SDFA A, trace $t \in E$ is either a possible trace according to the stochastic language of A, or is not. This distinction determines how the number of bits required to encode t is computed. If t is possible, then it is presumed to be encoded using the probability of t according to A, i.e., $L_A(t)$. Otherwise, t is presumed to be encoded using background knowledge about the log. These two modes are captured in this next definition.

Definition IV.1 (Trace compression cost): Let $t \in E$ be a trace in an event log $E \in \mathcal{E}$, let $A \in \mathcal{A}$ be an SDFA, and let $\mathit{bits} : \Lambda^* \times \mathcal{E} \times \mathcal{A} \to \mathbb{R}^+$ be a function that maps any and every sequence $t \in \Lambda^*$ to the number of bits required to uniquely encode t, given a viable encoding that assumes knowledge of E and A. The *trace compression cost of t in the presence of E and A* is defined:

$$\mathit{cost}_{\mathit{bits}}(t, E, A) := \begin{cases} -\log_2(L_A(t)) & t \in \hat{L_A} \\ \mathit{bits}(t, E, A) & \text{otherwise.} \end{cases}$$

⌟

For simplicity, in this first presentation we use a straightforward background model to measure $\mathit{bits}(t, E, A)$, ignoring both E and A, and trivially encoding t by taking each individual action as an equi-probable symbol over the underlying alphabet augmented by an "end of string" symbol, and with each trace terminated by that special symbol: $\mathit{bits}(t, E, A) := (1 + |t|) \log_2(1 + |\Lambda|)$. (We anticipate future exploration of more sophisticated encoding schemes that explore partial embeddings of traces into finite-context automata, and/or make use of the relative frequencies of the actions present in E.)

Given that definition, let A_1 and A_2 be SDFAs shown in Fig. 4 and Fig. 6, respectively. Then, based on logs E_1 and E_2 from Section III-C, the compression costs of traces $\texttt{abcd}$ and $\texttt{abce}$ are (in bits):

1) $\mathit{cost}_{\mathit{bits}}(\texttt{abcd}, E_1, A_1) = -\log_2(L_{A_1}(\texttt{abcd})) = -\log_2(1/32) = 5.00$;
2) $\mathit{cost}_{\mathit{bits}}(\texttt{abcd}, E_1, A_2) = -\log_2(L_{A_2}(\texttt{abcd})) = -\log_2(3/160) = 5.74$;
3) $\mathit{cost}_{\mathit{bits}}(\texttt{abce}, E_2, A_1) = (1 + |\texttt{abce}|) \log_2(1 + |\{\texttt{a},\texttt{b},\texttt{c},\texttt{d},\texttt{e}\}|) = 12.93$;

4) $cost_{bits}(\texttt{abce}, E_2, A_2) = -\log_2\left(L_{A_2}(\texttt{abce})\right) = -\log_2(3/64) = 4.42$.

Hence, A_1 compresses trace `abcd` better than does A_2, while A_2 compresses `abce` much better than A_1. In particular, `abce` is impossible according to A_1, and is encoded using the $bits(\cdot,\cdot,\cdot)$ function (case 3). That encoding needs five symbols: four actions and an "end of string" symbol, each taking $\log_2 6$ bits because the alphabet in E_2 contains five symbols, and an "end of string" symbol must also be included.

Also needed in the nominal compression cost is the "selector" associated with the diamond decision box in Fig. 2, and illustrated by the yellow line in Fig. 3. If the probability associated with a two-choice event is p, the expected cost of coding a stream of such choices is given by $H_0(p) := -p\log_2(p) - (1-p)\log_2(1-p)$; with, by definition, $H_0(0.0) := H_0(1.0) := 0.0$. These considerations lead directly to our main definition.

Definition IV.2 (Entropic relevance):
Let $E \in \mathcal{E}$ be an event log and let $A \in \mathcal{A}$ be an SDFA. Let $\rho(E, A)$ be the overall probability that a trace in E is possible in the stochastic language of A, $\rho(E, A) = \sum_{t \in \hat{L_A}} P(t \mid E)$. Then, the *entropic relevance of A to E*, or *relevance of A to E*, is denoted by $rel(E, A)$ and defined as:

$$rel(E, A) := H_0(\rho(E, A)) + \frac{1}{|E|}\sum_{t \in E} cost_{bits}(t, E, A)\,.$$

⌟

Fig. 2, presented earlier, explains Definition IV.1 and Definition IV.2. During the nominal encoding process, each trace t in the log is considered in turn. If t has a non-zero probability in the model, the corresponding selector code is generated (the left branch out of the decision diamond in Fig. 2), and then that probability is used to encode t relative to the model (the first option in Definition IV.1). If the probability of t is zero according to the model (the right branch in Fig. 2), the opposite selector code is needed, and then $bits(t, E, A)$ bits are used to represent t on a symbol-by-symbol basis in the universal background model (the second option in Definition IV.1). The selector codes associated with the diamond decision box, needed to differentiate between the two alternatives on a per-trace basis, add an average of $H_0(\rho(E, A))$ bits per trace.

Again, consider automata A_1 and A_2 from Fig. 4 and Fig. 6, respectively, and event logs E_1 and E_2 from Section III-C. Table I summarizes the constituent costs and the resulting entropic relevance values for the four combinations. In the first row, SDFA A_1 explains event log E_1 perfectly, as the traces in the log all fit the automaton, and the relative likelihoods of observing the traces in the log and in the automaton are identical. No other automaton can achieve lower relevance for E_1, and A_1 is *optimal*.

Automaton A_2 explains E_1 reasonably well, but with an increased model coding cost because of mis-matched probabilities for the empty trace and for the one-action trace `a` ($1/2$ vs $1/4$ for the empty trace, and $1/4$ vs $3/8$ for `a`).

The probability $\rho(E_2, A_1)$ that a trace from E_2 is possible according to L_{A_1} is 0.85, and hence $H_0(\rho(E, A)) = 0.61$ bits.

Autom.	Log	ρ	Select.	Bckgrd.	MdlCst.	Relevance
A_1	E_1	1.00	0.00	0.00	1.97	1.97
A_2	E_1	1.00	0.00	0.00	2.26	2.26
A_1	E_2	0.85	0.61	2.33	2.55	5.49
A_2	E_2	0.95	0.29	0.78	3.15	4.22

TABLE I: Entropic relevance (in bits) and its constituents for two example automata and two event logs. The columns list the probability of traces from the log being possible according to the automata (ρ); and the average (over all that log's traces) selector coding cost (Select.), background coding cost (Bckgrd.), and model coding cost (MdlCst.). The final column sums those three to get the entropic relevance.

That is, the choice between the background model and A_1 adds $-\log_2(\rho) = 0.23$ bits to the traces in E_2 that are possible according to L_{A_1}; adds $-\log_2(1-\rho) = 2.74$ bits for traces that are not possible; and averages at 0.61 bits per trace. The relevance of A_1 to E_2 is then obtained by adding in the arithmetic mean of the trace compression costs of the traces in E_2. The smallest compression cost arises for the empty trace (2.59 bits, part of the model coding cost), while the highest cost is associated with `abcdde` (18.09 bits, part of the background coding cost). Overall, A_1 can be used to compress E_2 using, on average, 5.49 bits per trace.

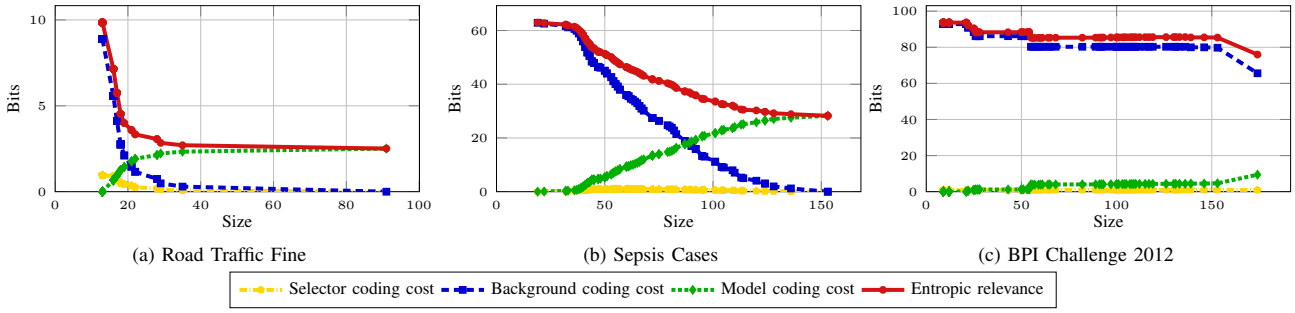
In the last row of the table, A_2 "explains" E_2 using 4.22 bits per trace, and hence A_2 is a better model for E_2 than is A_1. First, note that A_2 fits more of E_2's traces than does A_1, and with $\rho(E_2, A_1) = 0.85$ and $\rho(E_2, A_2) = 0.95$, the average cost of selecting between the process model and background model is less for A_2 than for A_1. In direct correspondence, the share of relevance stemming from the background coding cost is also less for A_2 than for A_1. Finally, despite the fact that more traces from E_2 fit A_2 than A_1, which pushes A_2's model coding cost higher, the overall cost of using A_2 is less.

V. Evaluation

To study the usefulness of entropic relevance, we used event logs from real-world IT-systems made publicly available by the IEEE Task Force on Process Mining.[1] For each log, directly follows models (DFMs, also known as DFGs) were constructed using the approach of Leemans et al. [6], with trace removal thresholds of $1/100$, $2/100$, …, $100/100$. Fig. 7 plots relevance values (and constituents) for three logs: Road Traffic Fine Management [7], Sepsis Cases [8], and BPI Challenge 2012 [9]. The *size* of each model (the horizontal axis) is taken to be the number of states plus the number of edges.

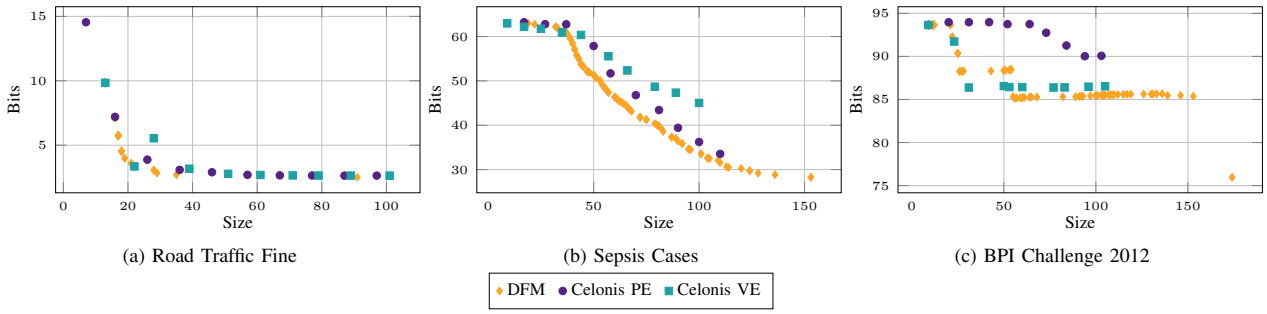
For the first two logs, the curves correspond to the anticipation provided by Fig. 3. In both cases as the models become more complex, an increasing fraction of traces fit the model (green line); a decreasing fraction are coded using the background model (blue line); and the selector coding cost (yellow line) is small over most of the range. Entropic relevance settles at around 2.5 bits per trace for Traffic Fines, indicating that the log is highly regular and hence highly compressible; and at around 30 bits per trace for Sepsis.

[1] https://data.4tu.nl/repository/collection:event_logs_real.



(a) Road Traffic Fine (b) Sepsis Cases (c) BPI Challenge 2012

Fig. 7: Entropic relevance and its constituents, plotted as a function of model size measured as states plus edges.



(a) Road Traffic Fine (b) Sepsis Cases (c) BPI Challenge 2012

Fig. 8: Entropic relevance, again plotted as a function of model size, for three different techniques.

In the third log in Fig. 7, a different picture emerges. Now the background model dominates the entropic cost and relevance values are high, a consequence of long traces that tend not to repeat in the log. Changing to a more nuanced background model – based on action occurrence frequencies and a zero-order model, for example – would lower the relevance values, but not decrease the model's reliance on them. In the context of the framework introduced here, the traces in this log are relatively inconsistent, and hence incompressible.

Celonis SE (https://www.celonis.com) granted us access to DFGs they constructed using "Celonis Snap", the free version of their enterprise-grade product. For each of the logs in Fig. 7, they generated twenty DFGs using two different techniques (PE and VE) and ten configurations for each technique. These DFGs were transferred to us on July 6, 2020. Fig. 8 plots relevance values in bits for these DFGs, again as a function of model size, plus the DFMs already used in Fig. 7. Now models, and hence methods, can be compared by considering the Pareto frontier of the points plotted in each graph. For these logs the relevance approach identifies the methods of Celonis SE as being preferable when the goal is to have small- to moderate-sized DFGs. However, note that the numeric relevance values are subject to the approach used to map DFGs to SDFAs and to the choice of background coding model. Detailed analysis of these interactions and of the DFGs discovered using other techniques is future work.

The computation of entropic relevance requires straightforward data structures and execution loops. With a hash-map used to implement the set of edges at each state in the model, computation time is linear in the total volume of log data processed (number of traces times average length). The average CPU time for computing relevance using our implementation on a commodity laptop computer for the largest analyzed log (BPI Challenge 2018 – Payment application; 43,809 traces and 984,613 events) over the 100 constructed DFMs (size ranged from 25 to 238) was 0.47 sec. Note also that none of the models plotted in Figs. 7 and 8 were over-fitted to the data, and the model description costs (see Fig. 3) were small compared to the entropic relevance scores. Our tool [10] and dataset [11] used to conduct the experiments are publicly available.

VI. Related Work

A plethora of non-stochastic process discovery and conformance checking techniques have been proposed over the last two decades, including the Genetic Mining [12], Heuristic Mining [13], Inductive Mining [14], and Split Mining [15] algorithms, all of which have been well-received by the process mining community. Non-stochastic conformance checking techniques can be broadly classified into *quantitative*, those that summarize conformance diagnostics into a single number, and *qualitative*, those that construct detailed analytics of commonalities and discrepancies between model and log traces. Carmona et al. [16] provide a useful overview.

Recently, Van der Aalst et al. [17], [18] initiated discussion of desired properties for conformance checking techniques. Entropy-based measures are the only quantitative conformance checking techniques that are known to satisfy all the properties for precision and recall that have been proposed to date [19], including the strict monotonicity properties.

The selection of currently available stochastic process mining techniques is rather scarce. To the best of our knowledge,

there are two stochastic discovery techniques proposed by academia. The technique presented by Rogge-Solti et al. [20] discovers stochastic Petri nets, while that of Leemans et al. [6] can be used to discover DFGs, and was employed in Section V. There are many commercial tools for discovering DFGs, or FDAGs, from event logs, but these are all closed source.

Two stochastic conformance checking techniques have been proposed. Leemans et al. [21] base their approach on the "earth movers' distance", and measure the effort to transform the distribution of log traces into the distribution of model traces, seen as two piles of dirt that need to be aligned with minimal effort. The technique is computationally demanding and suggests practical trade-offs between accuracy, run time, and memory usage. The approach of Leemans and Polyvyanyy [22] is inspired by entropy-based conformance checking [19]. Leemans and Polyvyanyy [22] also suggest a range of desired properties for stochastic precision and recall and show that their measures indeed possess these properties. The calculation of the measures requires (in the worst case) a quadratic number of steps in the size of the corresponding SDFAs, while entropic relevance runs in linear time in the size of the log. In addition, relevance, as defined here, reflects the compromise between precision and recall in a single value with meaningful units. Exploration of the relationships and correlations between these two previous measures and ours, and understanding of their differences, is an area for future work.

Finally, note that our proposal is an application of the *minimum description length* principle [23], [24]; which, in turn, is related to the 1965 definition by Kolmogorov that the intrinsic descriptive complexity of an object is the length of the shortest binary computer program that describes it [2]. Kolmogorov complexity formalizes the notion widely known as "Occam's Razor" [25], a problem-solving principle attributed to William of Ockham which suggests that the simplest (that is, shortest) sufficient explanation of a phenomenon is the best.

VII. Conclusion

We have presented an entropic relevance measure for stochastic conformance checking. The new measure is grounded in a minimum description length compression-based framework that assesses how accurately a stochastic process model describes an event log by computing the length of an encoding of the log traces relative to the stochastic language expressed by the model. The relevance of a model to a given log reflects a compromise between the precision and recall quality criteria, and is computable in time linear in the size of the log. Relevance is measured in bits, with values being directly interpretable, and with small scores being preferable.

Future work will investigate the effects of using different background models for calculating entropic relevance, with the aim of identifying models that lead to useful relevance measurements; noting that background model cost is one of the three components of the relevance calculation, and in some cases is dominant. We also plan to explore new techniques for discovering stochastic process models in a direct response to entropic relevance. Now that we have defined entropic relevance as a useful quantity, an important next step is to explicitly design models that seek to minimize it.

Acknowledgment. Artem Polyvyanyy was in part supported by the Australian Research Council project DP180102839. Hanan Alkhammash and Thomas Vogelgesang provided assistance with the preparation of the datasets used in Section V.

References

[1] W. M. P. van der Aalst, "A practitioner's guide to process mining: Limitations of the directly-follows graph," *Procedia Computer Science*, vol. 164, 2019.
[2] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed. Wiley, 2006.
[3] A. Moffat and A. Turpin, *Compression and Coding Algorithms*. Boston, MA: Kluwer Academic Publishers, Feb. 2002.
[4] R. C. Carrasco, "Accurate computation of the relative entropy between stochastic regular grammars," *ITA*, vol. 31, no. 5, 1997.
[5] W. M. P. van der Aalst, *Process Mining—Data Science in Action*, 2nd ed. Springer Berlin Heidelberg, 2016.
[6] S. J. J. Leemans, E. Poppe, and M. T. Wynn, "Directly follows-based process mining: Exploration & a case study," in *ICPM*. IEEE, 2019.
[7] M. De Leoni and F. Mannhardt, "Road traffic fine management process," 2015.
[8] F. Mannhardt, "Sepsis cases – event log," 2016.
[9] B. B. F. Van Dongen, "BPI challenge 2012," 2012.
[10] A. Polyvyanyy, H. Alkhammash, C. Di Ciccio, García-Bañuelos, A. Kalenkova, S. J. J. Leemans, J. Mendling, A. Moffat, and M. Weidlich, "Entropia: A Family of Entropy-Based Conformance Checking Measures for Process Mining," in *CoRR*, vol. abs/2008.09558, 2020.
[11] H. Alkhammash, A. Polyvyanyy, A. Moffat, and García-Bañuelos, "Discovered Process Models 2020-08," 8 2020. [Online]. Available: https://doi.org/10.26188/12814535
[12] 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, 2007.
[13] A. J. M. M. Weijters and W. M. P. van der Aalst, "Rediscovering workflow models from event-based data using little thumb," *Integrated Computer-Aided Engineering*, vol. 10, no. 2, 2003.
[14] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering block-structured process models from event logs - A constructive approach," in *Petri Nets 2013*, ser. LNCS, vol. 7927. Springer, 2013.
[15] A. Augusto, R. Conforti, M. Dumas, M. L. Rosa, and A. Polyvyanyy, "Split miner: automated discovery of accurate and simple business process models from event logs," *Knowl. Inf. Syst.*, vol. 59, no. 2, 2018.
[16] J. Carmona, B. F. van Dongen, A. Solti, and M. Weidlich, *Conformance Checking—Relating Processes and Models*. Springer, 2018.
[17] W. M. P. van der Aalst, "Relating process models and event logs—21 conformance propositions," in *ATAED*, ser. CEUR Workshop Proceedings, vol. 2115. CEUR-WS.org, 2018.
[18] A. F. Syring, N. Tax, and W. M. P. van der Aalst, "Evaluating conformance measures in process mining using conformance propositions," *Trans. Petri Nets Other Model. Concurr.*, vol. 14, 2019.
[19] A. Polyvyanyy, A. Solti, M. Weidlich, C. D. Ciccio, and J. Mendling, "Monotone precision and recall measures for comparing executions and specifications of dynamic systems," *ACM TOSEM*, vol. 29, no. 3, 2020.
[20] 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 Workshops*, ser. LNBIP, vol. 171, 2013.
[21] S. J. J. Leemans, A. F. Syring, and W. M. P. van der Aalst, "Earth movers' stochastic conformance checking," in *BPM Forum*, ser. LNBIP, vol. 360. Springer, 2019.
[22] S. J. J. Leemans and A. Polyvyanyy, "Stochastic-aware conformance checking: An entropy-based approach," in *CAiSE*, ser. LNCS, vol. 12127. Springer, 2020.
[23] A. Barron, J. Rissanen, and B. Yu, "The minimum description length principle in coding and modeling," *IEEE Trans. Inf. Theory*, vol. 44, no. 6, 1998.
[24] M. H. Hansen and B. Yu, "Model selection and the principle of minimum description length," *J. Am. Stat. Assoc.*, vol. 96, no. 454, 2001.
[25] S. C. Tornay, *Ockham: Studies and Selections*. La Salle, Ill., The Open Court Publishing Company, 1938.

Conformance Checking Approximation Using Simulation

Mohammadreza Fani Sani[1], Juan J. Garza Gonzalez[1], Sebastiaan J. van Zelst[2,1], and Wil M.P. van der Aalst[1,2]

1-Process and Data Science (PADS) Chair, RWTH-Aachen University, Germany
2-Fraunhofer FIT, Birlinghoven Castle, Sankt Augustin, Germany
Email:{fanisani, s.j.v.zelst, wvdaalst}@pads.rwth-aachen.de

***Abstract*—Conformance checking techniques are used to compute to what degree a process model and real execution data correspond to each other. In recent years, alignments have proven to be useful for calculating conformance statistics. Most alignment techniques provide an exact conformance value. However, in many applications, it suffices to have an approximated alignment value. Specifically, for large event data and using standard hardware, current alignment techniques are time-consuming and sometimes intractable. This paper proposes to use simulated behaviors of process models to approximate the conformance checking value. To simulate a process model, we exploit the behavior in the given event data. This method is independent from the process model notation and provides upper and lower bounds for the approximated alignment value. We assess the quality of our approximations and compare it to existing approximation techniques. The experiments on real event data show that using the proposed method, it is possible to achieve significant performance improvements.**

***Index Terms*—Process Mining, Conformance Checking Approximation, Alignment, Simulation, Edit Distance**

I. Introduction

Conformance checking aims to investigate the conformity of a discovered/designed process model w.r.t., real process executions [1]. Conformance checking techniques are used to detect deviations and to measure how accurate a process model is. It is possible to apply this branch of techniques to assess both event data and the process model. Alignments [2] were developed with the concrete goal to describe and quantify deviations in a non-ambiguous manner. Computing alignments has rapidly turned into the de facto standard conformance checking technique [3]. Moreover, alignments serve as a basis for other process mining methods that link event data to process models, e.g., they support performance analysis, decision mining [4], business process model repair [5], and prediction techniques. However, alignment computations may be time-consuming for real large event data.

In some scenarios, the diagnostic information that is produced by alignments is not required and we simply need an objective measure of model quality to compare process models, i.e., the alignment value. Moreover, in many applications, it is required to compute alignment values several times. For example, if we aim to discover an appropriate process model from event data, it is required to discover several process models using various process discovery algorithms with different settings, and, measure how each process model fits, w.r.t., the event data, i.e., by applying alignment techniques. As normal alignment methods take a considerable time for large real event data, analyzing many candidate process models is impractical. Therefore, by decreasing the alignment computation time, it is possible to consider more candidate process models in a limited time. Thus, by having an approximated conformance value, we are able to find a suitable process model faster. By providing bounds, we guarantee that the accurate alignment value does not exceed a range of values, and, consequently we determine if it is required to do further analysis or not, which saves a considerable amount of time. Thus, it is valuable to have a quick approximated conformance value and it is excellent worth to let users adjust the level of approximation.

It is shown in [6] that using the edit distance function, a subset of model behaviors can be used instead of the process model for conformance approximation. It suggests to sample some behaviors from the event log and approximate their alignments. In this paper, we extend the previous work by proposing to use process model simulation (i.e., some of its possible executable behaviors) to create a subset of process model behaviors. The core idea of this paper is to have simulated behaviors close to the recorded behaviors in the event log. Moreover, we provide bounds for the actual conformance value. Using the proposed method, users are able to adjust the amount of process model behaviors considered in the approximation, which affects the computation time and the accuracy of alignment values and their bounds. As the proposed method just uses the simulated behaviors for conformance approximation, it is independent of any process model notation. Because we use the edit distance function and do not compute any alignment, even if there is no reference process model and just some of the correct behaviors of the process (e.g., some of the valid variants) are known, the proposed method is able to approximate the conformance value. The method additionally returns problematic activities, based on their deviation rates.

We implemented the proposed method using both the ProM [7] and RapidProM [8] platforms. Moreover, we applied it to several large real event data and process models. We also compared our approach with the state-of-the-art alignment approximation method [6]. The results show that the proposed simulation method improves the performance of the conformance checking process while providing approximations close to the actual values.

978-1-7281-9832-3/20/$31.00 ©2020 IEEE
DOI 10.1109/ICPM49681.2020.00025

The remainder of this paper is structured as follows. In Section II, we discuss related work. Section III defines preliminary notation. We explain the proposed method in Section IV and evaluate it in Section V. Section VI concludes the paper and presents some future work.

II. Related Work

In this section, we explain some conformance checking and simulation techniques in the process mining domain. For a complete overview of conformance checking techniques in process mining, we refer to [9] and [10]. Early work in conformance checking uses token-based replay [11]. This technique replays a trace of executed events from the event log on a Petri net and adds missing tokens if transitions are not able to fire. After the replay phase, the conformance statistic is computed based on remaining and missing tokens. Alignments were introduced in [12] and have rapidly developed into the standard conformance checking technique [3]. In [13] and [14], decomposition techniques are proposed for alignment computation. Applying decomposition techniques generally improves computation time. These techniques use the divide-and-conquer paradigm. However, these techniques are primarily beneficial when there are lots of unique activities in the process [15]. The authors in [3] and [16] also propose to incrementally compute prefix-alignments, and providing real-time conformance checking for event data streams. Recently, a general stochastic conformance checking method is proposed in [17] which requires a stochastic process model that is not available in many cases.

Some research has been done to approximate the alignment value. [18] uses deep learning to approximate alignment statistics. Moreover, a recursive general approach for approximating the alignment, i.e., computation of near-optimal alignments, has been proposed in [19]. Moreover, [20] uses a statistical trace sampling approach that approximates the conformance value without proving bounds for the actual alignment value. The authors of [21] suggest a conformance approximation method that applies relaxation labeling methods to a partial order representation of a process model. Similar to the previous method, it does not provide any guarantee for the approximated value. Furthermore, it needs to preprocess the process model each time. Finally, [6] recommends using a subset of model behaviors for conformance approximation and bounds for the actual value. It suggests applying the alignment technique for some traces to build the subset of model behaviors. In this paper, we propose a guided simulation method to build the subset of model behavior to generate behaviors that are closer to recorded behaviors in the event data. Unlike many conformance checking methods, this method is independent of process model notation and considers a process model as a set of possible behaviors.

Different approaches to simulation in process mining have been proposed. These approaches are mostly at an instance level, i.e., a detailed level such as the framework presented in [22], which uses the process detailed information to create a CPN model for simulation. Moreover, [23] recommends an approach to simulate an event log based on a given (stochastic) process model. In [24], a simulation approach to simulate business process models that uses information in event logs is proposed. Another direction of the simulation in process mining is an aggregated level simulation, e.g., [25], in which the system dynamics modeling technique is introduced for "what-if" analysis in processes.

III. Preliminaries

In this section, we briefly introduce basic process mining and, specifically, conformance checking terminology and notations that ease the readability of this paper.

Given a set X, a multiset B over X is a function $B\colon X\rightarrow\mathbb{N}_{\geq 0}$ that allows certain elements of X to appear multiple times. We write a multiset as $B=[e_1^{k_1}, e_2^{k_2}, ..., e_n^{k_n}]$, where for $1\leq i\leq n$, we have $B(e_i)=k_i$ with $k_i\in\mathbb{N}_{\geq 0}$. If $k_i=1$, we omit its superscript, and if for some $e\in X$ we have $B(e)=0$, we omit it from the multiset notation. $\overline{B}=\{e\in X|B(e)>0\}$ is the set of unique elements present in the multiset. The set of all multisets over a set X is written as $\mathcal{B}(X)$.

A sequence of length n over members of set X is a function $\sigma:\{1,2,...,n\}\rightarrow X$ which defines the occurrence order of elements of set X. We show a sequence using the notation $\sigma=\langle s_1,...,s_n\rangle$ where $s_i=\sigma(i)$, for $1\leq i\leq n$. We denote with $s_i\in\sigma$ that s_i is an element of the sequence σ. Moreover, the set of all possible sequences over set X is shown by X^*. The empty sequence is denoted with ϵ. Furthermore, $|\sigma|$ indicates the length of sequence σ, e.g., $|\langle a,d,d,e\rangle|=4$. Function $hd\colon X^*\times\mathbb{N}_{\geq 0}\rightarrow X^*$, returns the "head" of a sequence, i.e., given a sequence $\sigma\in X^*$, $hd(\sigma,k)=\langle s_1,s_2,..,s_k\rangle$, i.e., the sequence of the first k elements of σ. If $k\geq|\sigma|$, then $hd(\sigma,k)=\sigma$. Symmetrically, $tl\colon X^*\times\mathbb{N}_{\geq 0}\rightarrow X^*$ returns the "tail" of a sequence and is defined as $tl(\sigma,k)=\langle s_{n-k+1},s_{n-k+2},...,s_n\rangle$, i.e., the sequence of the last k elements of σ. If $k\geq|\sigma|$, $tl(\sigma,k)=\sigma$. We define that $hd(\sigma,0)=tl(\sigma,0)=\epsilon$. The concatenation of two sequences σ_1 and σ_2 with length m and n is sequence $\sigma_3=\sigma_1\cdot\sigma_2$ with length $m+n$ where $\sigma_3(i)=\sigma_1(i)$ for $1\leq i\leq m$ and $\sigma_3(i)=\sigma_2(i-m)$ for $m+1\leq i\leq m+n$. A subsequence σ_s of σ is obtained by removing $|\sigma|-|\sigma_s|$ elements of σ where the result equals σ_s. For example, $\langle b,d\rangle$ is a subsequence of $\langle a,b,c,d,e\rangle$. Moreover, σ' is a strict subsequence of sequence σ if there exist σ_1 and σ_2 such that $\sigma=\sigma_1\cdot\sigma'\cdot\sigma_2$ and we denote it by $\sigma'\sqsubset\sigma$; furthermore, if $\sigma_1=\epsilon$, we say that σ' is a prefix of σ.

Having two sequences, function $\omega:X^*\times X^*\rightarrow X^*$ returns one of the longest common subsequence of them. For example, $\omega(\langle a,b,d,e\rangle,\langle c,a,d,f,e\rangle)=\langle a,d,e\rangle$. Finally, $fq:X^*\times X^*\rightarrow\mathbb{N}_{\geq 0}$ is a function that returns how many times in a sequence a subsequence is present. For example, $fq(\langle d,a,d,e\rangle,\langle d\rangle)=2$.

Event logs are the starting point of many process mining algorithms. For alignment computation, we just use the control-flow information of the event logs. Therefore, we define an event log as follows.

*Definition 1 (**Event Log**):* Let $\mathcal{A}$ be the universe of activities and $A\subseteq\mathcal{A}$ is a set of activities. An event log is a multiset of

sequences over A, i.e., $L \in \mathcal{B}(A^*)$. Moreover, we refer to each $\sigma \in \overline{L}$ as a "*variant*" whereas $L(\sigma)$ denotes how many traces of the form σ are presented within the event log. Moreover, $|\overline{L}|$ refers to the number of variants in the event log.
For example, in the event log that is presented in Figure 1, $A=\{a,b,c,d,e\}$, $|\overline{L}|=5$, and $L(\langle a,b,e\rangle)=2$.

There are many notations to describe the possible behaviors described by a process model. Here, we define a process model using a set of all its possible replayable behaviors as follows.

*Definition 2 (**Process Model**):* Let $\mathcal{A}$ be the universe of activities and $A \subseteq \mathcal{A}$ is a set of activities. A process model is a set of sequences over A, i.e., $M \subseteq A^*$.
For example, in the process model that is presented in Figure 1 in a Petri net notation, we have $\langle a,b,d,b,c,e\rangle \in M$, $\langle a,b,e\rangle \in M$, and $\langle a,b,d,b,e\rangle \in M$. Note that in case of having an unbounded loop in a process model, Set M is infinite.

As we define event logs and process models as collections of sequences of activities, we are able to use edit distance and the optimal alignment functions to compute their alignment.

*Definition 3 (**Edit Distance and Optimal Alignment Cost**):* Let A be a set of activities and $\sigma_l, \sigma_m \in A^*$ be two traces. $\triangle: A^* \times A^* \to \mathbb{N}_{\geq 0}$ is a function such that $\triangle(\sigma_l,\sigma_m)=|\sigma_l|+|\sigma_m|-2\times|\omega(\sigma_l,\sigma_m)|$ returns the minimal number of required edits to transform σ_l to σ_m. In addition, we define $\delta: A^* \times A^* \to B(A) \times B(A)$ that returns two multisets of synchronous and asynchronous moves. Synchronous moves correspond to one of the longest common subsequences of the given sequences and asynchronous moves represents the edited activities that are required to transform subsequences. Moreover, $\Gamma: A^* \times \mathbb{P}(A^*) \to \mathbb{N}_{\geq 0}$ is a function that returns the minimum number of edits to transfer a trace to one of the traces in a set of traces, i.e., the optimal alignment cost.
For example, $\triangle(\langle a,b,d,b,e\rangle, \langle a,c,d,e\rangle)=3$ that corresponds two deletions (i.e., b) and one insertion (i.e., c). Therefore, $\delta(\langle a,b,d,b,e\rangle, \langle a,c,d,e\rangle)=([a,d,e],[b^2,c])$. By using $\restriction_1$ and $\restriction_2$, we retrieve the synchronous and asynchronous moves respectively. For example, $([a,d,e],[b^2,c^1])\restriction_2=[b^2,c]$ are the asynchronous moves. Note that, there may exist more than one longest common subsequence for two sequences and the δ is non-deterministic. Moreover, for the model that is presented in Figure 1, $\Gamma(\langle a,c,e\rangle, M)=1$ that corresponds to transfer $\langle a,c,e\rangle$ to $\langle a,b,c,e\rangle \in M$ or $\langle a,c,b,e\rangle \in M$. In [6], it is shown that $\Gamma(\sigma, M)$ equal to the optimal alignment cost.

To compute the fitness value of a trace and a model we use the following equation.

$$fitness_{Trace}(\sigma_L, M) = \frac{\Gamma(\sigma_L, M)}{|\sigma_L| + M_s} \tag{1}$$

In the above equation, $M_s = \min_{\sigma \in M}(|\sigma|)$ is the length of the shortest trace in the model. To compute the fitness value of an event log and a process model, we use the following formula.

$$Fitness(L, M) = \frac{\sum_{\sigma \in \overline{L}} L(\sigma) \times fitness_{Trace}(\sigma, M)}{\sum_{\sigma \in \overline{L}} L(\sigma)} \tag{2}$$

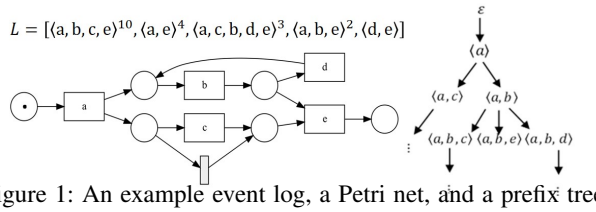


Figure 1: An example event log, a Petri net, and a prefix tree.

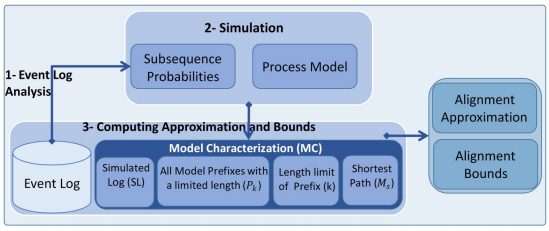


Figure 2: A schematic view of the proposed method.

IV. Approximating Alignments using Subset of Model Behavior

In this section, we present the proposed conformance approximation method. We aim to approximate the alignment value without any alignment computation. A schematic view of this method is shown in Figure 2. We first analyze the event log; then, based on the probabilities of subsequences, we provide characteristics of the process model using the simulation method. Afterwards, based on these characteristics, we provide prefixes for the alignment approximation. In the following, each stage is explained in more details.

A. Event log analysis

In this stage, we traverse the event log to find all the activities and variants in the event log. Moreover, we compute the occurrence probability of different sequences with a specific length. These probabilities guide the simulation algorithm to generate model traces close to the available traces in the event log. The occurrence probability of a subsequence $\sigma' \in A^*$ is computed as follows.

$$Prob(\sigma', L) = \frac{\sum_{\sigma \in \overline{L}} L(\sigma) \times fq(\sigma, \sigma')}{\sum_{\sigma \in \overline{L}} \left(\sum_{\sigma'' \sqsubseteq \sigma \wedge |\sigma''| = |\sigma'|} (L(\sigma) \times fq(\sigma, \sigma'')) \right)} \tag{3}$$

For example, for the event log that is presented in Figure 1, $Prob(\langle a,b,c\rangle, L)=\frac{10}{10+10+3+3+3+2}$ or $Prob(\langle a,b,c,e\rangle, L)=\frac{10}{10+3+3}$. By getting the maximum length of subsequence from the user, we are able to compute the probabilities of them.

B. Simulation

The input of the simulation algorithm is a process model and an event log and it provides a *Model Characterization* that is defined as follows.

*Definition 4 (**Model Characterization**):* Let $M \subseteq A^*$ be a process model, a 4-tuple $MC=(SL, P_k, k, M_s)$ be a model characterization of M, where $SL \subseteq M$ is a subset of model traces and $M_s = \min_{\sigma \in M}(|\sigma|)$ is the length of the shortest trace in model M. Moreover, $P_k=\{hd(\sigma,i) | \sigma \in M \wedge i \leq k\}$ is the set of all prefixes in M with length less or equal to k.

To simulate the process model, we use a prefix tree. In Figure 1 a process model and a part of its prefix tree are shown. We consider sequence σ' as a prefix of model M if $\exists_{\sigma \in M \wedge l \in \mathbb{N}_{\geq 0}}(hd(\sigma, l){=}\sigma')$. We use P_k to show the information of the prefix tree that contains all the observed prefixes.

Most process model notations, e.g., BPMN and Petri net, allow us to clarify which activities are executable after the execution of sequence of activities (i.e., a prefix). Thus, we define prefix extension as follows.

*Definition 5 (**Prefix extension**):* Let A be a set of activities and let $\sigma' {\in} A^*$ be a sequence of activities and let $M {\subseteq} A^*$ be a process model. The prefix extension function $\alpha{:}A^* {\times} \mathbb{P}(A^*) {\nrightarrow} \mathbb{P}(A^*)$ returns the set of all possible extensions of prefix σ'. In other words, $\alpha(\sigma', M){=}\{\sigma' {\cdot} \langle a \rangle | \exists_{\sigma \in M} (\sigma' {\cdot} \langle a \rangle {=} hd(\sigma, |\sigma'|{+}1)\}$.
For example, if M is corresponding to the process model that is presented in Figure 1, $\alpha(\langle a, b \rangle, M){=}\{\langle a, b, c \rangle, \langle a, b, d \rangle, \langle a, b, e \rangle\}$.

The simulation algorithm starts with initializing $P{=}\{\epsilon\}$ and $SL{=}\{\}$. At each step, we select one of the non-extended prefixes in P and discover all its possible extensions, i.e., $\alpha(\sigma', M)$, and append them to P. Moreover, if any of the newly extended prefixes are complete trace, we append them to SL. Therefore, the updated sets are $P{=}P {\cup} (\alpha(\sigma', M))$ and $SL{=}SL {\cup} \{\sigma {\in} \alpha(\sigma', M) | \sigma {\in} M\}$. Note that a prefix $\sigma' {\in} P$ is non-extended if $\nexists_{\sigma" \in P \setminus \{\sigma'\}} (\sigma' {=} hd(\sigma", |\sigma'|))$.

The simulation method is guided to select the prefix with the highest probability according to Equation 3. In this regard, for all non-extended prefixes in P, the method computes $Prob(tl(\sigma', l), L)$, where l indicates the maximum length of subsequences that is given by the user. For example, in Figure 1, if $P{=}\{\epsilon, \langle a \rangle, \langle a, b \rangle, \langle a, c \rangle\}$, we know that $\langle a, b \rangle$ and $\langle a, c \rangle$ are non-extended prefixes. Therefore, the method chooses $\langle a, b \rangle$ which has a higher probability in the event log. By extending this prefix, we have $P{=}\{\epsilon, \langle a \rangle, \langle a, b \rangle, \langle a, c \rangle\} {\cup} \{\langle a, b, c \rangle, \langle a, b, d \rangle, \langle a, b, e \rangle\}$, and $SL{=}\{\langle a, b, e \rangle\}$.

By having the k value, we guarantee that all possible prefixes of the model with length less or equal to k are present in P_k. The proposed simulation method finds k based on the length of the shortest non-extended prefix in P.

We continue the simulation procedure until at least one of the following conditions is satisfied: 1) the number of simulated traces, i.e., $|SL|$ reaches to the simulation size, i.e., given by the user 2) there exist no non-extended prefix in P which means we could not simulate any new traces 3) having $k {\geq} 2 {\times} \max_{\sigma \in \overline{L}}(|\sigma|) {+} M_s$ that indicates all the unseen prefixes are longer than two times of the longest trace in the event log plus the shortest path in the model. In the next subsection, we prove that if we have the third condition, we are able to find the optimal alignment cost using the simulated traces. Moreover, we use P_k notation to show a set of all observed prefixes with length less or equal to k where $P_k{=}\{\sigma' {\in} P \mid |\sigma'| {\leq} k\}$.

To sum up, the output of the simulation stage is $MC{=}(SL, P_k, k, M_s)$. Note that any other simulation method that generates a model characterization can be used in the proposed approximation method. For example, it is possible to extend the prefix tree in a breadth first traversal order. If the simulation method could not guarantee to have a set of complete prefixes with a specific length, we should use $P_k{=}\{\epsilon\}$ and $k{=}0$ in the corresponding MC.

C. Computing Alignment Cost Bounds and Approximation

In this stage, for each variant $\sigma {\in} \overline{L}$, we compute the bounds and an approximation for its alignment cost based on a model characterization $MC{=}(SL, P_k, k, M_s)$ which can be produced by any simulation method. Thus, the proposed approach is independent of the process model notation and the simulation method to provide bounds and approximation values. Here, we first explain how to compute the upper and lower bounds; then, the approximation method is described.

Upper bound: For the upper bound of the alignment cost, we compute $UB(MC, \sigma){=}\Gamma(\sigma, SL)$. It is shown in [6] that $\Gamma(\sigma, SL) {\geq} \Gamma(\sigma, M)$ as Γ returns the distance of the most similar trace and $SL {\subseteq} M$. In the following, we prove that if $k {\geq} 2{\times}|\sigma|{+}M_s$, then $\Gamma(\sigma, SL)$ returns the optimal alignment. Note that by having P_k, SL contains all the model traces with length less or equal to k.

*Lemma 1 (**Maximum required length of prefixes**):* We know that $\Gamma(\sigma, M) {\leq} |\sigma| {+} M_s$. Suppose that $\sigma'_m {\in} M$ corresponds to the optimal alignment of σ, i.e., $\Gamma(\sigma, M){=}\triangle(\sigma, \sigma'_m)$. Therefore, $\triangle(\sigma, \sigma'_m) {\leq} |\sigma| {+} M_s$ and according to Definition 3, $|\sigma| {+} |\sigma'_m| {-} 2 {\times} |\omega(\sigma, \sigma'_m)| {\leq} |\sigma| {+} M_s$. Note that the maximum length of the longest common subsequence of two sequences is at most the length of the shorter sequence. Thus, in the worst case, $|\sigma'_m| {\leq} 2{\times}|\sigma| {+} M_s$. Therefore, the length of the model trace corresponds to the optimal alignment of trace σ should be less or equals to $2{\times}|\sigma| {+} M_s$. In other words, if $k {\geq} 2 {\times} \max_{\sigma \in \overline{L}}(|\sigma|) {+} M_s$, we return the actual fitness value using SL.

Lower bound: For the lower bound, we use the maximum value of three values, i.e., $LB(MC, \sigma){=}Max(lb_1, lb_2, lb_3)$. let $A' {\subseteq} A$ be the set of all activities in the model, $lb_1(MC, \sigma){=}|\sigma| {-} |\sigma {\restriction}_{A'}|$ where $\sigma {\restriction}_{A'}$ is a projection of sequence σ on activities in set A'. It means if we have some activities in the trace that are not present in the model, we can easily consider them as asynchronous moves. Moreover, as discussed in [6], $lb_2(MC, \sigma){=}\max(M_s {-} |\sigma {\restriction}_{A'}|, 0)$ is a lower bound for the optimal alignment cost that is useful if the length of the trace is shorter than the shortest path in the model. Note that both lb_1 and lb_2 are independent from the simulation result. Finally, we compute the third lower bound as $lb_3(MC, \sigma){=}\Gamma(hd(\sigma, k), P_k)$. As we have all possible prefixes of model M with length less or equal to k in P_k, lb_3 computes the minimum number of edits that is required to have a valid prefix for trace σ. As $\epsilon {\in} P_k$, we always have $0 {\leq} lb_3(MC, \sigma) {\leq} \min(k, |\sigma|)$. All lb_1, lb_2 and lb_3 are valid lower bounds for the actual alignment cost; therefore, we use the highest number to have a tighter bound.

Approximation: In both SL and $\overline{L}$, because of the presence of loops in the process, it is possible that a subsequence is

Table I: Result of using the proposed approximation method for the event log that is given in Figure 1, using model characterization $MC=(\{\langle a,b,e\rangle\},\{\epsilon,\langle a\rangle,\langle a,b\rangle,\langle a,c\rangle\},2,3\}$.

Trace/ Event Log	$\Gamma(\sigma,M)$	$\Gamma(\sigma,SL)$	Actual Fitness	Lower Bound Fitness	Upper Bound Fitness	Approximated Fitness	Frequency
$\langle a,b,c,e\rangle$	0	1	1.0	0.857	1.0	0.857	10
$\langle a,e\rangle$	1	1	0.8	0.8	0.8	0.8	4
$\langle a,c,b,d,e\rangle$	1	2	0.875	0.75	1	0.75	3
$\langle a,b,e\rangle$	0	0	1	1	1	1	2
$\langle d,e\rangle$	3	3	0.6	0.4	0.6	0.4	1
L	$\sim$	$\sim$	0.921	0.821	0.94	0.821	$\sim$

present several times in a trace. The repetitive patterns increase the number of unique behaviors which leads to inaccurate approximations considering a constant MC. We define the repetitive patterns of a sequence as follows.

*Definition 6 (**Repetitive Patterns**):* Let $\sigma\in X^*$ be a sequence. Given a non-empty strict subsequence $\sigma'\sqsubseteq\sigma$, we call σ' a *repetitive pattern* if $\sigma'\cdot\sigma'\sqsubseteq\sigma$. Moreover, we define function $\lambda{:}X^*\rightarrow\mathbb{P}(X^*)$ that receives a sequence and returns the set of all *repetitive patterns* in it.

For instance, we have $\lambda(\langle a,d,d,d,d,e\rangle)=\{\langle d\rangle,\langle d,d\rangle\}$ and $\lambda(\langle a,f,g,f,g,e\rangle)=\{\langle f,g\rangle\}$.

To deal with the problem of the existence of repetitive patterns, we propose to compress sequences and keep only one repetition of repetitive patterns. The sequence compression function is defined as follows.

*Definition 7 (**Sequence Compression**):* Let σ' be a *repetitive pattern* of σ and $\sigma=\sigma_1\cdot\sigma'..\sigma'\cdot\sigma_2$. We say that σ_c is a *compression* of σ by σ' if $\sigma_c=\sigma_1\cdot\sigma'\cdot\sigma_2\cdot$. Moreover, we define *sequence compression* function $\theta{:}(X^*\backslash\{\epsilon\})\times(X^*\backslash\{\epsilon\})\rightarrow X^*$ is a function such that $\theta(\sigma,\sigma')=\sigma_c$.

Therefore, if $\theta(\sigma,\sigma')=\sigma_c$, then $\sigma'\notin\lambda(\sigma_c)$. For example, $\theta(\langle a,d,d,d,d,e\rangle,\langle d\rangle)=\langle a,d,e\rangle$.

To approximate the alignment cost, we consider both original and compressed traces of the simulated log and the original event log. Therefore, we compute $Apx(MC,\sigma)=\min_{\sigma"\in\{\theta(\sigma,\sigma')|\sigma'\in\lambda(\sigma)\}}(\Gamma(\sigma",SL\cup SL_c))$. However, by compressing the traces, it is possible to remove some asynchronous moves which causes to have approximated alignment cost smaller than the lower bound. Therefore, if the computed $Apx(MC,\sigma)<LB(MC,\sigma)$, we use $Apx(MC,\sigma)=\frac{LB(MC,\sigma)+UB(MC,\sigma)}{2}$.

To compute the fitness bounds and approximation for traces, UB, Apx, and LB values should be used instead of $\Gamma(\sigma,M)$ in Equation 1, e.g., $fitness_{LB}(MC,\sigma)=\frac{UB(MC,\sigma)}{|\sigma|+M_s}$. To compute the lower bound for the fitness we need to use UB and to have the fitness upper bound, LB should be used. Moreover, to have fitness bounds and approximation for an event log and a model characterization, we use the weighted average of values for the variants in the event log similar to Equation 2, e.g., $Fitness_{Apx}(L,MC)=\frac{\sum_{\sigma\in L}L(\sigma)\times fitness_{APX}(MC,\sigma)}{\sum_{\sigma\in L}L(\sigma)}$.

In Table I, the computed bounds and the approximated fitness value for each variant and the overall event log of Figure 1 is given based on having only one simulated trace and k=2, i.e., $MC=(\{\langle a,b,e\rangle\},\{\epsilon,\langle a\rangle,\langle a,b\rangle,\langle a,c\rangle\},2,3\}$. The approximated fitness is 0.821 and the proposed fitness bounds are 0.94 and 0.821. By increasing $|SL|$, we expect to have more accurate approximations and bounds. Furthermore, using the δ function, we find the number of asynchronous moves for each activity which is 1 for *a*, 2 for *b*, *c*, and *d*, and 0 for *e*.

Table II: The real event logs that are used in the experiment. Artificial *start* and *end* activities are inserted in all the traces.

Event Log	**Activities#**	**Traces#**	**Variants#**
BPIC-2012 [26]	25	13087	4336
BPIC-2017 *[27]*	28	31509	15930
BPIC-2018-*Inspection* [28]	17	5485	3190
BPIC-2019 [29]	44	251734	11973
Hospital-Billing [30]	20	100000	1020
RTFM [31]	13	150370	231
Sepsis [32]	18	1050	846

V. Evaluation

In this section, we aim to explore the accuracy and the performance of the proposed method. We first explain the implementation and the experimental setting. Thereafter, the experimental results and a discussion of results are provided.

A. Implementation

To apply the proposed conformance approximation method, we implemented the *Conformance Approximation using Simulation* plug-in in the `ProM` framework[1] [7]. It takes an event log and a process model as inputs and returns the conformance approximation, its bounds, and the deviation rates of different activities. In this implementation, we consider a Petri net representation. However, other notations can be supported using our proposed method. The given Petri net can have silent transitions or duplicate labels, however, it should be sound. The user is able to adjust the maximum number of the simulated traces. Another parameter of this plug-in is the maximum length of subsequence in computation of probabilities according to Equation 3.

To apply the method on various event logs with different parameters, we ported the developed plug-in to `RapidProM`, i.e., an extension of `RapidMiner` and combines scientific work-flows with a several process mining algorithms [8].

B. Experimental Setup

We applied the proposed methods on seven different real event logs. Some information about these logs is given in Table II. To discover models, we used the Inductive miner [33] with infrequent thresholds equal to 0.2, 0.3, 0.5, and 0.7.

In the first experiment, we compared the proposed method with the *sampling* method [6] that generates a subset of model traces by alignment computation of some traces in the event log. Here, we use its default setting, i.e., computing alignment

[1]svn.win.tue.nl/repos/prom/Packages/LogFiltering and http://www.promtools.org/doku.php

Table III: Comparison of approximating the conformance checking using the proposed simulation method and the sampling method that is proposed in [6]. For the sampling method, we selected the 10% of the most frequent variants in the event logs and for the simulation method, we used $|SL|$ equal to the generated model traces by the sampling method.

log	Process Model (IMi)	|SL|	Actual Fitness	Normal Alignment time (ms)	Approximation Error		Bound Width		PI		SL Computation Time (ms)	
					Sampling	Simulation	Sampling	Simulation	Sampling	Simulation	Sampling	Simulation
BPIC 2012	0.2	395	0.953	74173	**0.020**	0.061	**0.07**	0.19	5.8	**46.6**	8280	**574**
	0.3	262	0.848	41200	0.049	**0.022**	**0.09**	0.16	4.9	**37.2**	5398	**347**
	0.5	18	0.641	22823	0.160	**0.124**	**0.14**	0.19	7.9	**90.3**	2180	**111**
	0.7	12	0.620	19946	0.158	**0.121**	**0.15**	0.23	8.6	**88.6**	1662	**97**
BPIC 2017	0.2	1453	0.933	6826957	**0.014**	0.077	**0.09**	0.33	19.5	**491.1**	210399	**2724**
	0.3	813	0.841	6150221	**0.008**	0.115	**0.11**	0.41	21.9	**737.0**	195192	**1513**
	0.5	6	0.344	139041	0.041	**0.009**	**0.33**	0.63	2.9	**108.6**	30383	**854**
	0.7	6	0.344	195562	0.041	**0.000**	**0.33**	0.64	2.4	**153.3**	46257	**849**
BPIC 2018 Inspection	0.2	150	0.819	1633669	**0.016**	0.026	**0.16**	0.42	48.1	**1786.0**	23397	**401**
	0.3	132	0.780	855934	0.016	**0.014**	**0.17**	0.44	35.4	**775.3**	13451	**665**
	0.5	78	0.476	64064	0.090	**0.018**	**0.35**	0.53	13.9	**78.8**	2426	**264**
	0.7	76	0.469	13360	0.090	**0.012**	**0.36**	0.53	3.4	**17.2**	1837	**253**
BPIC 2019	0.2	656	0.937	58316184	**0.004**	0.051	**0.02**	0.08	33.4	**2662.1**	1733189	**12635**
	0.3	132	0.864	5399054	**0.003**	0.015	**0.02**	0.15	152.3	**1239.9**	26054	**1367**
	0.5	44	0.827	999677	0.008	**0.002**	**0.03**	0.16	36.9	**215.7**	20675	**1116**
	0.7	145	0.844	1911284	0.008	**0.006**	**0.02**	0.16	77.1	**562.5**	17563	**1858**
Hospital	0.2	29	0.956	7054	0.002	**0.001**	**0.01**	0.04	1.5	**14.8**	4622	**409**
	0.3	26	0.925	7460	0.002	**0.005**	**0.01**	0.05	2.2	**17.1**	3292	**355**
	0.5	6	0.907	4042	0.002	**0.000**	**0.01**	0.03	1.1	**10.2**	3639	**307**
	0.7	7	0.907	4075	0.002	**0.001**	**0.01**	0.03	1.2	**10.0**	3335	**346**
RTFM	0.2	9	0.989	1392	**0.001**	0.040	**0.00**	0.05	0.4	**3.1**	3889	**445**
	0.3	9	0.967	1015	**0.000**	0.003	**0.00**	0.04	0.3	**1.7**	3932	**590**
	0.5	14	0.869	1004	**0.000**	0.001	**0.00**	0.04	0.3	**2.5**	3846	**399**
	0.7	6	0.932	957	0.001	**0.000**	**0.00**	0.07	0.3	**2.4**	3765	**399**
Sepsis	0.2	76	0.921	12528	**0.009**	0.100	**0.14**	0.20	5.2	**67.2**	2173	**54**
	0.3	22	0.613	5169	0.081	**0.005**	**0.32**	0.34	9.4	**129.2**	480	**21**
	0.5	20	0.586	3762	**0.011**	0.019	**0.32**	0.38	8.0	**74.0**	402	**23**
	0.7	18	0.586	3693	**0.011**	0.018	**0.32**	0.38	7.8	**74.3**	398	**20**

of 10% of the most frequent variants. However, the generated model traces using this method could be less than the number of computed alignments. Therefore, we use the simulation method with $|SL|$ equal to the number of unique model behaviors that are generated by sampling method.

In the second experiment, we analyse the effect of changing the simulation parameters on the accuracy and performance of the proposed method. Therefore, we used the simulation method with $|SL|$ equals to $100, 500,$ and 1000 and the subsequence length equals $1, 2, 3,$ and 4. Moreover, in the last experiment, we compared the proposed simulation method, with a random simulation method. In this regard, we used the subsequence length equal to 2 and $|SL|$ equal to $10, 50, 100, 500, 1000,$ and 10000 for Sepsis event log.

In all the experiments and for all methods, we used one thread of CPU. Moreover, each experiment was repeated four times, since the conformance checking and simulation times are not deterministic, and the average values are shown. For computing the normal conformance checking, we used the method that is proposed in [34].

To evaluate whether the proposed simulation method is able to improve the performance of the conformance checking process, we measure $PI = \frac{\text{Normal Conformance Time}}{\text{Approximated Conformance Time}}$. In this formula, a higher PI value means conformance is computed in less time. As both approximation methods need a preprocessing phase (e.g., traversing the event log and computing a subset of model behaviors), we compute PI considering the preprocessing time.

The approximation error, i.e., the difference between approximated fitness value and the actual fitness value shows how the accuracy of approximation that is computed by $ApproxErr = |ActualFitness - ApproxFitness|$. Also, we measure the bound width of an approximation by computing $BoundWidth = UBFitness - LBFitness$. A tighter bound width means we have more accurate bounds.

C. Experimental Result and Discussion

In Table III, we show how different approximation methods improve the performance of conformance checking. For both sampling and simulation methods, in most of the cases, we have improvement in the performance of conformance checking (i.e., $PI > 1$). This improvement for the simulation method is much higher in all the the cases. It is mainly because, the required time to generate model traces is less than using alignments. When we have complex process models and large event logs, the improvement is higher. However, in these cases, we have a higher approximation error and the provided bounds are not accurate enough. However, the provided bounds for the conformance value are always worse when the simulation method is used. It is because, the sampling method knows the exact conformance values of the 10% of the most frequent variants. Due to a Pareto-line distribution a small number of variants may account for a large number of traces.

Table IV: Analysing the effect of the simulation size on the approximation time and the accuracy results. Here, the average value are shown when the subsequence length for computing probabilities equals to 2 and IMi=0.2.

Event Log	|SL|	Simulation Time (ms)	Approximation Time (ms)	PI	Approximation Error	Bound Width
BPIC 2019	10	40	2173	6409	0.046	0.164
	100	598	4313	3129	0.028	0.148
	1000	5400	14214	931	0.016	0.136
BPIC 2018 Inspection	10	6	393	16	0.002	0.038
	100	225	640	10	0.000	0.020
	1000	437	881	8	0.000	0.020
Hospital	10	3	376	1881	0.069	0.546
	100	218	860	876	0.018	0.475
	1000	2384	3291	196	0.004	0.426
RTFM	10	1	509	2	0.008	0.046
	100	18	491	2	0.000	0.021
	1000	230	692	2	0.000	0.011
Sepsis	10	2	46	138	0.061	0.351
	100	26	97	63	0.025	0.313
	1000	208	341	23	0.012	0.208

The result shows that we are able to have more accurate approximations using the simulation method specifically when we have more precise process models (that are discovered using a higher threshold in the Inductive miner) even with just few simulated traces. Note that the corresponding model behavior for alignment of several variants may be similar. Therefore, the sampling method usually generates fewer model traces compared to the number of computed alignments. For example, for RFTM event log and process model that is discovered using IMi=0.7, it computed alignments of 24 variants to generate 6 model traces.

For some logs, none of the methods are able to provide accurate bounds (with the applied setting). Specifically, when the process model is imprecise, the output of the proposed method is not accurate. The simulation of an imprecise process model usually generates many behaviors that may be far from variants that are in the event log. Thus, the provided bounds are far from each other which is a limitation of our method. To have more accurate results in these cases, we should simulate more traces that decrease the performance improvement.

In the next experiment, we analyze the effect of the number of simulated traces on the simulation and approximation times and the accuracy of the approximation. In this experiment, we considered subsequences in Equation 3 equals to 2 and used the process models with threshold equals to $0, 2$.

The results of this experiment are shown in Table IV. The results indicate that increasing the size of $|SL|$ increases the required time for simulation and finding the distance of the most similar trace (i.e., the Approximation time). Results show that for some event logs which have simple structured process models, e.g., *RTFM*, even with 100 simulated traces, we are able to detect accurate conformance value. Moreover, for some event logs, it is not possible to generate the specified number of unique model traces, which is the reason that PI does not decrease as expected. Also, by increasing the size of the simulated traces, we provide more accurate approximations and bounds. Therefore, the user is able to trade-off between the performance improvement and accuracy of the approximated value by adjusting this parameter. We also conducted a similar experiment for analyzing the effects of changing the length of subsequences when we compute probabilities. We found that this parameter has no significant impact on the performance of the proposed method. However, when $|SL|$ is low, the length of subsequence affects the accuracy of approximation value and the provided bounds.

Table V: Comparison of using the proposed simulation method with random simulation technique [6] for Sepsis event log when the Inductive miner with threshold 0.2 is used.

	Approximation Error		Bound Width		Total Approximation Time (ms)	
|SL|	Random Simulation	Proposed Method	Random Simulation	Proposed Method	Random Simulation	Proposed Method
10	0.408	0.186	0.437	0.308	75	82
50	0.304	0.122	0.332	0.224	104	164
100	0.284	0.104	0.312	0.214	122	219
500	0.235	0.065	0.263	0.152	197	573
1000	0.219	0.053	0.248	0.133	355	1010
10000	0.171	0.023	0.200	0.092	1805	8846

Finally, we analyzed the effect of considering log behaviors in the simulation process on the accuracy of provided conformance approximation and its bounds. As the base line, we used the *random* simulation method that was originally proposed by [35] and used for conformance checking approximation in [6]. The results of this experiment are presented in Table V which show that the proposed method detects more accurate approximation and bounds. Also, the random sampling method is faster than our method. It is because there is no need to have a probability computation in this method and also it may generate traces that are already presented in SL.

VI. Conclusion

In this paper, we proposed an approximation method for computing conformance values including providing upper and lower bounds based on process model simulation. We consider a process model as a set of all possible behaviors than can be executed by the process. This assumption allows us to obtain conformance checking results even for the cases that we do not have a descriptive process model. Using the simulation method, we generate a subset of the model's behaviors. We guide the simulation method to generate traces that are more similar to the recorded behaviors in the event log. We apply these simulated traces for approximating the conformance value using the edit distance function. Moreover, the method provides lower and upper bounds for the approximated value based on the seen simulated behaviors.

To evaluate the proposed method, we implemented our technique using both ProM and RapidProM and applied our implementation to seven real event logs and 28 discovered

process models. The results show that the proposed method is able to decrease the conformance checking computation time and simultaneously find approximated values close to the actual alignment value. We found that when the process model is imprecise, the accuracy of the approximation degrades. Furthermore, experiments show that considering behaviors in the event log in simulation improves the accuracy of the approximation value and bounds compared to the case that a process model is simulated randomly.

As future work, we aim to provide a platform in which gives us qualitative feedback in a faster way. Moreover, it is possible to provide an incremental approximation tool that increases the number of simulated traces incrementally and lets the end-user decide when the accuracy is enough.

ACKNOWLEDGMENT

We thank the Alexander von Humboldt (AvH) stiftung for funding this research.

REFERENCES

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

[2] A. Adriansyah, J. Munoz-Gama, J. Carmona, B. van Dongen, and W. M. P. van der Aalst, "Alignment based Precision Checking," in *International Conference on Business Process Management*. Springer, 2012, pp. 137–149.

[3] S. J. van Zelst, A. Bolt, M. Hassani, B. F. van Dongen, and W. M. van der Aalst, "Online conformance checking: relating event streams to process models using prefix-alignments," *International Journal of Data Science and Analytics*, pp. 1–16, 2017.

[4] M. De Leoni and W. M. van der Aalst, "Data-aware process mining: discovering decisions in processes using alignments," in *Proceedings of the 28th annual ACM symposium on applied computing*. ACM, 2013, pp. 1454–1461.

[5] D. Fahland and W. M. P. van der Aalst, "Model Repair—Aligning Process Models to Reality," *Information Systems*, vol. 47, pp. 220–243, 2015.

[6] M. Fani Sani, S. J. van Zelst, and W. M. P. van der Aalst, "Conformance Checking Approximation Using Subset Selection and Edit Distance," in *Advanced Information Systems Engineering - 32nd International Conference, CAiSE 2020, Grenoble, France, June 8-12, 2020, Proceedings*, 2020, pp. 234–251. [Online]. Available: https://doi.org/10.1007/978-3-030-49435-3_15

[7] W. M. P. van der Aalst, B. van Dongen, C. W. Günther, A. Rozinat, E. Verbeek, and T. Weijters, "ProM: The Process Mining Toolkit," *BPM (Demos)*, vol. 489, no. 31, 2009.

[8] W. M. P. van der Aalst, A. Bolt, and S. van Zelst, "RapidProM: Mine Your Processes and Not Just Your Data," *CoRR*, vol. abs/1703.03740, 2017.

[9] M. Elhagaly, K. Drvoderić, R. G. Kippers, and F. A. Bukhsh, "Evolution of Compliance Checking in Process Mining Discipline," in *2019 2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*. IEEE, 2019, pp. 1–6.

[10] J. Carmona, B. van Dongen, A. Solti, and M. Weidlich, *Conformance Checking*. Springer, 2018.

[11] A. Rozinat and W. M. van der Aalst, "Conformance checking of processes based on monitoring real behavior," *Information Systems*, vol. 33, no. 1, pp. 64–95, 2008.

[12] 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 Interdiscip. Rev. Data Min. Knowl. Discov.*, vol. 2, no. 2, pp. 182–192, 2012. [Online]. Available: https://doi.org/10.1002/widm.1045

[13] W. M. van der Aalst, "Decomposing Petri nets for process mining: A generic approach," *Distributed and Parallel Databases*, vol. 31, no. 4, pp. 471–507, 2013.

[14] J. Munoz-Gama, J. Carmona, and W. M. van der Aalst, "Single-entry single-exit decomposed conformance checking," *Information Systems*, vol. 46, pp. 102–122, 2014.

[15] H. M. W. Verbeek, W. M. P. van der Aalst, and J. Munoz-Gama, "Divide and Conquer: A Tool Framework for Supporting Decomposed Discovery in Process Mining," *Comput. J.*, vol. 60, no. 11, pp. 1649–1674, 2017. [Online]. Available: https://doi.org/10.1093/comjnl/bxx040

[16] D. Schuster and S. J. van Zelst, "Online Process Monitoring Using Incremental State-Space Expansion: An Exact Algorithm," in *BPM 2020, Proceedings*, ser. Lecture Notes in Computer Science. Springer, 2020. [Online]. Available: https://sebastiaanvanzelst.com/wp-content/uploads/2020/06/2020_bpm_schuster_zelst_incremental_alignments.pdf

[17] S. J. J. Leemans and A. Polyvyanyy, "stochastic-aware conformance checking: An entropy-based approach."

[18] T. Nolle, A. Seeliger, N. Thoma, and M. Mühlhäuser, "Deepalign: Alignment-based process anomaly correction using recurrent neural networks," in *International Conference on Advanced Information Systems Engineering*. Springer, 2020, pp. 319–333.

[19] F. Taymouri and J. Carmona, "A recursive paradigm for aligning observed behavior of large structured process models," in *International Conference on Business Process Management*. Springer, 2016, pp. 197–214.

[20] M. Bauer, H. van der Aa, and M. Weidlich, "Estimating Process Conformance by Trace Sampling and Result Approximation," pp. 179–197, 2019.

[21] L. Padró and J. Carmona, "Approximate Computation of Alignments of Business Processes Through Relaxation Labelling," in *International Conference on Business Process Management*. Springer, 2019, pp. 250–267.

[22] A. Rozinat, M. T. Wynn, W. M. P. van der Aalst, A. H. M. ter Hofstede, and C. J. Fidge, "Workflow simulation for operational decision support," *Data Knowl. Eng.*, vol. 68, no. 9, pp. 834–850, 2009. [Online]. Available: https://doi.org/10.1016/j.datak.2009.02.014

[23] A. Rogge-Solti, R. S. Mans, W. M. P. van der Aalst, and M. Weske, "Improving Documentation by Repairing Event Logs," in *IFIP Working Conference on The Practice of Enterprise Modeling*. Springer, 2013, pp. 129–144.

[24] M. Camargo, M. Dumas, and O. G. Rojas, "Automated discovery of business process simulation models from event logs," vol. abs/1910.05404, 2019. [Online]. Available: http://arxiv.org/abs/1910.05404

[25] M. Pourbafrani, S. J. van Zelst, and W. M. P. van der Aalst, "Scenario-Based Prediction of Business Processes Using System Dynamics," in *On the Move to Meaningful Internet Systems: OTM 2019 Conferences - Confederated International Conferences: CoopIS, ODBASE, C&TC 2019, Rhodes, Greece, October 21-25, 2019, Proceedings*, 2019, pp. 422–439. [Online]. Available: https://doi.org/10.1007/978-3-030-33246-4_27

[26] Van Dongen, B.F. (Boudewijn), "BPI Challenge 2012," 2012. [Online]. Available: https://data.4tu.nl/repository/uuid:3926db30-f712-4394-aebc-75976070e91f

[27] Van Dongen, B.F., "BPIC 2017. Eindhoven University of Technology," 2017. [Online]. Available: https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b

[28] Van Dongen, B.F. (Boudewijn) and Borchert, F. (Florian), "BPI Challenge 2018," 2018. [Online]. Available: https://data.4tu.nl/repository/uuid:3301445f-95e8-4ff0-98a4-901f1f204972

[29] Van Dongen, B.F. (Boudewijn), "BPI Challenge 2019," 2019. [Online]. Available: https://data.4tu.nl/repository/uuid:d06aff4b-79f0-45e6-8ec8-e19730c248f1

[30] F. Mannhardt, "Hospital billing-event log," *Eindhoven University of Technology. Dataset*, pp. 326–347, 2017.

[31] M. De Leoni and F. Mannhardt, "Road traffic fine management process," *Eindhoven University of Technology. Dataset*, 2015.

[32] F. Mannhardt, "Sepsis cases-event log. Eindhoven University of Technology," 2016. [Online]. Available: https://doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460

[33] S. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour," in *BPI*, 2014, pp. 66–78.

[34] B. F. van Dongen, "Efficiently Computing Alignments - Algorithm and Datastructures," in *Business Process Management Workshops - BPM 2018 International Workshops, Sydney, NSW, Australia, September 9-14, 2018, Revised Papers*, 2018, pp. 44–55.

[35] A. Rogge-Solti and M. Weske, "Prediction of business process durations using non-Markovian stochastic Petri nets," vol. 54, 2015, pp. 1–14. [Online]. Available: https://doi.org/10.1016/j.is.2015.04.004

A Temporal Logic-Based Measurement Framework for Process Mining

Alessio Cecconi
WU Vienna
Vienna, Austria
alessio.cecconi@wu.ac.at

Giuseppe De Giacomo, Claudio Di Ciccio
Sapienza University of Rome
Rome, Italy
{name.surname}@uniroma1.it

Fabrizio Maria Maggi
Free University of Bolzano
Bolzano, Italy
maggi@inf.unibz.it

Jan Mendling
WU Vienna
Vienna, Austria
jan.mendling@wu.ac.at

***Abstract*—The assessment of behavioral rules with respect to a given dataset is key in several research areas, including declarative process mining, association rule mining, and specification mining. The assessment is required to check how well a set of discovered rules describes the input data, as well as to determine to what extent data complies with predefined rules. In declarative process mining, in particular, some measures have been taken from association rule mining and adapted to support the assessment of temporal rules on event logs. Among them, support and confidence are used most often, yet they are reportedly unable to provide a sufficiently rich feedback to users and often cause spurious rules to be discovered from logs. In addition, these measures are designed to work on a predefined set of rules, thus lacking generality and extensibility. In this paper, we address this research gap by developing a general measurement framework for temporal rules based on Linear-time Temporal Logic with Past on Finite Traces (LTLp$_f$). The framework is independent from the rule-specification language of choice and allows users to define new measures. We show that our framework can seamlessly adapt well-known measures of the association rule mining field to declarative process mining. Also, we test our software prototype implementing the framework on synthetic and real-world data, and investigate the properties characterizing those measures in the context of process analysis.**

***Index Terms*—Declarative Process Mining, Specification Mining, Association Rule Mining, Quality Measures, Temporal Rules**

I. Introduction

Measuring the degree to which process traces comply with behavioral rules is key in process analysis branches such as conformance checking [1], compliance assessment [2], and discovery of process constraints [3]. To date, several measures have been defined to this end. Among the most frequently used measures there are support and confidence. However, their definition has been customized to the specification languages in use and even for the specific mining algorithms under analysis. For instance, there is a significant difference in the definition of support used in [3] (percentage of traces fully compliant to a rule) and [4] (percentage of fulfilled triggers of the rule over the entire log), in a way that the support of rule "If a is executed, then b will be executed later" on a set of traces like $\{\langle \mathsf{a},\mathsf{b},\mathsf{c},\mathsf{d}\rangle, \langle \mathsf{a},\mathsf{b},\mathsf{c},\mathsf{a}\rangle, \langle \mathsf{a},\mathsf{c}\rangle\}$ is equal to 0.33 for [3] and 0.5 according to [4]. Furthermore, the definition of those measures are defined ad hoc for specific sets of rules, like DECLARE [5] templates. Such issues hinder the fair comparison and eventually the advancement of rule-based process mining.

A plethora of other measures are available in the context of association rule mining that are reportedly superior in comparison to support and confidence [6]. However, these measures do not take into consideration the temporal dimension, which is a first-class citizen dimension in process mining.

In this paper, we address this research problem by proposing a general measurement framework rooted in formal semantics, specifically in Linear-time Temporal Logic with Past on Finite Traces (LTLp$_f$), to express process rules in a reactive form [7] and abstract from specific rule-specification languages. More specifically, we show that a fine grained temporal logic interpretation of any formula in the form "if A then B" allows us to assess all the available association rule mining measures as-is for temporal rules.

Towards investigating the implications of using a large set of available measures on log analysis, we conduct an extensive set of simulation experiments. Through them, we observe that the measures respond differently to changes in the behavior evidenced by event logs, thus suggesting that different measures can be used to highlight different aspects of a process as per its recorded executions. Also, we test our implementation of the framework on both synthetic and real-world data.

The remainder of this paper is structured as follows. Section II discusses prior research on measures for declarative process mining and specification mining. Section III defines preliminaries upon which we define our framework. Section IV defines the measurement framework. Section V presents computational studies of our implementation and Section VI the results of its application to a series of simulation experiments. Finally, Section VII summarizes the contribution of this paper and points to opportunities for future research.

II. Related Work

Temporal logic and declarative specifications have been widely used to support process discovery and conformance checking. The assessment of rules with respect to traces is a key component of all these techniques.

In declarative process discovery, quality measures are used to prune candidate rules based on user-defined thresholds. This pruning is used for DECLARE discovery in [3], [4] and for DCRgraphs discovery in [8]. These techniques are mainly based on confidence and support measures. Nevertheless, the use of support and confidence is not sufficient to avoid a large

978-1-7281-9832-3/20/$31.00 ©2020 IEEE
DOI 10.1109/ICPM49681.2020.00026

number of spurious results, thus threatening their statistical validity [9]. In addition, the definitions of these metrics are also different for different techniques. For instance, the support measure presented in [3] is different from the support of [4]. Also, both are defined for the sole DECLARE template set.

In the area of conformance checking, de Leoni et al. [1] use alignments and fitness to measure the degree of conformance of a DECLARE model with respect to an execution trace. Polyvyanyy et al. [10] rely on entropy to measure precision and recall of both procedural and declarative models. Finally, Burattin et al. [11] use specific measures like fulfillment ratio and violation ratio based on the evaluation of the number of activations of a rule that lead to a fulfillment and the number of activations that lead to a violation in a log. Quality measures are critical for all these techniques. However, the metrics provided in these works are bound to the specific techniques and the precise modeling languages used. Also, those techniques do not provide a general measurement framework that can be applied for any type of temporal-logic rule.

In software engineering, a number of works employ techniques for the discovery and quality assessment of temporal patterns. This stream of research is called specification mining. Yang et al. [12] discover 2-value temporal patterns using a trace measure that quantifies partial satisfactions of a rule. Yet, the technique lacks generality as it is limited only to alternation patterns (similar to ALTERNATERESPONSE and ALTERNATEPRECEDENCE in Table I) and the adopted computation heuristics are tailored to the software domain. Le et al. [6] emphasize the limits of using only support and confidence measures and investigate properties of other measures reviewed in [13]. Their results demonstrate that there are several measures outperforming support and confidence, and that the combination of different measures yields better results. However, they limit their study to 2-value temporal patterns (specifically, RESPONSE and PRECEDENCE in Table I). Furthermore, their computation of the probability for a temporal specification is based on a sliding window technique [14]: the traces are read in chunks of the size of a given window, then the probability of a rule is the percentage of windows in which it is valid. They test the effect of different window sizes, showing that their results depend not only on the input rules and the data, but also on this parameter selection. Lemieux et al. [15] extend specification mining to arbitrary Linear Temporal Logic (LTL) specifications (implicitly on finite traces) beyond 2-value templates. Yet, they resort only on support and confidence measures to prune uninteresting results, thus incurring in the already mentioned statistical limits [9].

In summary, we observe that measures are largely used as a tool to support research, but they are hardly made subject to evaluative research themselves. Next, we define preliminaries for a framework to assess various measures.

III. PRELIMINARIES

As the formal foundations of our framework, we consider rules specified in Linear Temporal Logic on Finite Traces (LTL_f) [16], as used in DECLARE [5], [17]. LTL_f has exactly the same syntax of LTL [18]. Its semantics is interpreted on finite traces, and hence takes into account that business processes are assumed to terminate sooner or later [19]. DECLARE focuses on some specific LTL_f formulas. Table I illustrates some of the most important rules for business process modeling.

Table I: Some DECLARE constraints expressed as RCons

Constraint	LTL_f expression [16]	RCon
PARTICIPATION(a)	$\Diamond \mathsf{a}$	$t_{\text{Start}} \boxdot\!\!\rightarrow \Diamond \mathsf{a}$
INIT(a)	a	$t_{\text{Start}} \boxdot\!\!\rightarrow \mathsf{a}$
END(a)	$\Box\Diamond \mathsf{a}$	$t_{\text{End}} \boxdot\!\!\rightarrow \mathsf{a}$
ATMOSTONE(a)	$\Box(\mathsf{a} \rightarrow \bigcirc(\neg\Diamond \mathsf{a}))$	$\mathsf{a} \boxdot\!\!\rightarrow \bigcirc(\neg\Diamond \mathsf{a})$
RESPONDEDEXISTENCE(a, b)	$\Diamond \mathsf{a} \rightarrow \Diamond \mathsf{b}$	$\mathsf{a} \boxdot\!\!\rightarrow (\Diamond \mathsf{b} \vee \Diamond\!\!-\!\mathsf{b})$
RESPONSE(a, b)	$\Box(\mathsf{a} \rightarrow \Diamond \mathsf{b})$	$\mathsf{a} \boxdot\!\!\rightarrow \Diamond \mathsf{b}$
ALTERNATERESPONSE(a, b)	$\Box(\mathsf{a} \rightarrow \Diamond \mathsf{b}) \wedge \Box(\mathsf{a} \rightarrow \bigcirc(\neg \mathsf{a}\ \mathbf{W}\ \mathsf{b}))$	$\mathsf{a} \boxdot\!\!\rightarrow \bigcirc(\neg \mathsf{a}\ \mathbf{U}\ \mathsf{b})$
CHAINRESPONSE(a, b)	$\Box(\mathsf{a} \rightarrow \Diamond \mathsf{b}) \wedge \Box(\mathsf{a} \rightarrow \bigcirc \mathsf{b})$	$\mathsf{a} \boxdot\!\!\rightarrow \bigcirc \mathsf{b}$
PRECEDENCE(a, b)	$\neg \mathsf{b}\ \mathbf{W}\ \mathsf{a}$	$\mathsf{b} \boxdot\!\!\rightarrow \Diamond\!\!-\!\mathsf{a}$
ALTERNATEPRECEDENCE(a, b)	$(\neg \mathsf{b}\ \mathbf{W}\ \mathsf{a}) \wedge \Box(\mathsf{b} \rightarrow \bigcirc(\neg \mathsf{b}\ \mathbf{W}\ \mathsf{a}))$	$\mathsf{b} \boxdot\!\!\rightarrow \ominus(\neg \mathsf{b}\ \mathbf{S}\ \mathsf{a})$
CHAINPRECEDENCE(a, b)	$(\neg \mathsf{b}\ \mathbf{W}\ \mathsf{a}) \wedge \Box(\bigcirc \mathsf{b} \rightarrow \mathsf{a})$	$\mathsf{b} \boxdot\!\!\rightarrow \ominus \mathsf{a}$

Specifically useful for our purposes is LTLp_f, which is an extension of LTL_f supporting the expression of properties of the past (hence the "p" suffix) [7]. Well-formed LTLp_f formulae are built from an alphabet $\Sigma \supseteq \{\mathsf{a}\}$ of propositional symbols and are closed under the boolean connectives, the unary temporal operators $\bigcirc$ (next) and $\ominus$ (previous), the binary temporal operators $\mathbf{U}$ (Until) and $\mathbf{S}$ (Since):

$$\varphi ::= \mathsf{a} | (\neg\varphi) | (\varphi_1 \wedge \varphi_2) | (\bigcirc\varphi) | (\varphi_1\ \mathbf{U}\ \varphi_2) | (\ominus\varphi) | (\varphi_1\ \mathbf{S}\ \varphi_2).$$

From these basic operators, it is possible to derive: classical boolean abbreviations *True*, *False*, $\vee$, $\rightarrow$; constant t_{End}, verified as $\neg \bigcirc \mathit{True}$, denoting the last instant of a trace; constant t_{Start}, verified as $\neg \ominus \mathit{True}$, denoting the first instant of a trace; $\Diamond\varphi$ as $\mathit{True}\ \mathbf{U}\ \varphi$ indicating that φ holds true eventually before t_{End}; $\varphi_1\ \mathbf{W}\ \varphi_2$ as $(\varphi_1\ \mathbf{U}\ \varphi_2) \vee \Box\varphi_1$, which relaxes $\mathbf{U}$ as φ_2 may never hold true; $\Diamond\!\!-\!\varphi$ as $\mathit{True}\ \mathbf{S}\ \varphi$ indicating that φ holds true eventually in the past after t_{Start}; $\Box\varphi$ as $\neg\Diamond\neg\varphi$ indicating that φ holds true from the current instant till t_{End}; $\boxminus\varphi$ as $\neg\Diamond\!\!-\!\neg\varphi$ indicating that φ holds true from t_{Start} to the current instant.

Given a finite trace t of length $n \in \mathbb{N}$, an LTLp_f formula φ is satisfied in a given instant i ($1 \leqslant i \leqslant n$) by induction of the following:
$t, i \models \mathit{True}$; $t, i \not\models \mathit{False}$;
$t, i \models \mathsf{a}$ iff $t(i)$ is assigned with a;
$t, i \models \neg\varphi$ iff $t, i \not\models \varphi$;
$t, i \models \varphi_1 \wedge \varphi_2$ iff $t, i \models \varphi_1$ and $t, i \models \varphi_2$;
$t, i \models \bigcirc\varphi$ iff $i < n$ and $t, i+1 \models \varphi$;
$t, i \models \ominus\varphi$ iff $i > 1$ and $t, i-1 \models \varphi$;
$t, i \models \varphi_1\ \mathbf{U}\ \varphi_2$ iff $t, j \models \varphi_2$ with $i \leqslant j \leqslant n$, and $t, k \models \varphi_1$ for all k s.t. $i \leqslant k < j$;
$t, i \models \varphi_1\ \mathbf{S}\ \varphi_2$ iff $t, j \models \varphi_2$ with $1 \leqslant j \leqslant i$, and $t, k \models \varphi_1$ for all k s.t. $j < k \leqslant i$.
A formula φ is satisfied by a trace t, written $t \models \varphi$ iff $t, 1 \models \varphi$. One of the central properties of LTLp_f, shared with LTL_f, is that one can compute a deterministic finite state automaton (DFS) A_φ such that for every trace t we have $t \models \varphi$ iff t is in the language recognized by A_φ.

IV. Temporal-extended Measurement Framework

In this section, we build on LTLp$_f$ to develop our measurement framework. Above, we discussed that relying only on confidence and support measures can produce spurious results [9] and that various measures beyond them have been proposed in association rule mining for time-unaware rules [13]. Table II presents a comprehensive list of those measures. Some of them, like confidence and support, are used for the temporal specification of processes, though interpreted in different, ad-hoc ways [3], [4], [7]. The framework we propose next is generic as it allows for the usage of any probabilistic measure (including those of Table II) on any rule for the temporal specification of processes. To this end, Section IV-A formalizes the reactive temporal specification of rules, Section IV-B discusses their probabilistic interpretation, and Section IV-C defines the overall framework.

A. Reactive Temporal Specification

Our first building block is the concept of Reactive Constraint (RCon), originally introduced in [7], which we extend here. A rule typically expresses that the occurrence of certain preconditions (activation) implies certain consequences (target). We codify such intuition in RCons based on LTLp$_f$.

Definition 4.1 (Reactive Constraint (RCon)): Given an alphabet $\Sigma \cup \{t_{\text{Start}}, t_{\text{End}}, \mathit{True}, \mathit{False}\}$, let φ_α and φ_τ be LTLp$_f$ formulae over Σ. A *Reactive Constraint (RCon)* Ψ is a pair $(\varphi_\alpha, \varphi_\tau)$ hereafter denoted as $\Psi \triangleq \varphi_\alpha \boxdot\!\!\rightarrow \varphi_\tau$.

An RCon is interpreted as follows: each time the activator is true, the target should be true at that point of the trace. For example, $\mathsf{a} \boxdot\!\!\rightarrow \Diamond \mathsf{b}$ is an RCon describing that every time a (the activator, φ_α) is *True*, then also $\Diamond \mathsf{b}$ (the target, φ_τ) must evaluate to *True*. Because at every event of the trace (i.e., any point in time) both the activator and target can be either *True* or *False*, the possible evaluation of an RCon can result in either of the following four combinations.

Definition 4.2 (RCon evaluation): Given an RCon $\Psi \triangleq \varphi_\alpha \boxdot\!\!\rightarrow \varphi_\tau$ and a trace t of length $n \in \mathbb{N}$, let t_i be the i^{th} event in the trace ($1 \leqslant i \leqslant n$). For each $t_i \in t$ the possible evaluations of Ψ are:

$\varphi_\alpha = \mathit{False}, \varphi_\tau = \mathit{False}$ if $\{t, i \not\models \varphi_\alpha \wedge t, i \not\models \varphi_\tau\}$;
$\varphi_\alpha = \mathit{False}, \varphi_\tau = \mathit{True}$ if $\{t, i \not\models \varphi_\alpha \wedge t, i \models \varphi_\tau\}$;
$\varphi_\alpha = \mathit{True}, \varphi_\tau = \mathit{False}$ if $\{t, i \models \varphi_\alpha \wedge t, i \not\models \varphi_\tau\}$;
$\varphi_\alpha = \mathit{True}, \varphi_\tau = \mathit{True}$ if $\{t, i \models \varphi_\alpha \wedge t, i \models \varphi_\tau\}$.

For example, the first two rows of Table III show the evaluation of RCon $(\Diamond \mathsf{b} \wedge \Diamond \mathsf{e}) \boxdot\!\!\rightarrow (\neg \mathsf{c} \vee \Diamond \mathsf{f})$ in each event of trace $\langle \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{f}, \mathsf{c}, \mathsf{e}, \mathsf{c}, \mathsf{h} \rangle$. The constraint states that, between the execution of tasks b and e, no occurrence of c is expected unless it is eventually followed by f. Notice that φ_α and φ_τ are evaluated separately at every event of a trace.

The RCon evaluation can be performed efficiently based on the automaton-based techniques defined in [7], adapting it for offline verification. The full description of this aspect goes beyond the scope of the paper, but we briefly outline the rationale here. Intuitively, we resort on [7, Theorem 4]: an RCon can be separated in pure-past, pure-present and pure-future components. The respective sub-formulas contain only past temporal operators, none, or only future ones. Therefore, by mirroring pure-past formulas and reversing their automata, a single replay of the sub-trace from the beginning to the activator event keeps track of the truth value of the pure-past formula till that point. As we know the suffix of the trace, we can apply the same principle to pure-future formulas too: a single replay from the end of the trace to the activator event keeps track of the truth value of the pure-future formula from that point *onwards*. In this way, we evaluate any formula at each event reading the trace only twice: once from t_{Start} to t_{End} (past components) and once from t_{End} to t_{Start} (future components). This implies that the computational cost depends linearly on the number of events in the event log and the number of rules to verify. Specifically, given an event log L with $|L|$ traces, assuming every trace $t \in L$ having length up to n events, and M rules, the cost to verify all rules on L is: $O(|L| \times n \times M)$.

B. Probabilistic interpretation

The evaluation of RCons indicates whether a rule holds true or false within a trace. In real life, traces often contain noise or partially deviate from desired process specifications. In such occasions wherein the trace contains also events that do not satisfy the rule, we are interested in understanding *to what degree* a rule is satisfied. As we have previously defined the notion of satisfaction for φ_α and φ_τ on single events (Def. 4.2), we can devise a probabilistic interpretation for RCons over traces.

Definition 4.3 (Probability of an LTLp$_f$ formula): Given an LTLp$_f$ formula φ and a trace t of length $|t| = n$, we define the probability of φ over $t = \langle t_1, \ldots, t_n \rangle$ as the proportion of events t_i ($i \in [1, n], n \in \mathbb{N}$) satisfying φ.

$$P(\varphi, t) = \frac{|\{i \in [1, n] : t, i \models \varphi\}|}{n}$$

Definition 4.4 (Probability of LTLp$_f$ formulae intersection): Given two LTLp$_f$ formulae φ_1 and φ_2 and a trace t of length n, we define the probability of the intersection of φ_1 and φ_2 over t as the proportion of events $t_i \in t$ satisfying both φ_1 and φ_2.

$$P(\varphi_1 \cap \varphi_2, t) = \frac{|\{i \in [1, n] : t, i \models \varphi_1, t, i \models \varphi_2\}|}{n}$$

The probability of all the RCon evaluations follows from the above definitions.

$$\begin{cases} P(\neg\varphi_\alpha \cap \neg\varphi_\tau, t) = \dfrac{|\{i \in [1, n] : t, i \not\models \varphi_\alpha, t, i \not\models \varphi_\tau\}|}{n} \\ P(\neg\varphi_\alpha \cap \varphi_\tau, t) = \dfrac{|\{i \in [1, n] : t, i \not\models \varphi_\alpha, t, i \models \varphi_\tau\}|}{n} \\ P(\varphi_\alpha \cap \neg\varphi_\tau, t) = \dfrac{|\{i \in [1, n] : t, i \models \varphi_\alpha, t, i \not\models \varphi_\tau\}|}{n} \\ P(\varphi_\alpha \cap \varphi_\tau, t) = \dfrac{|\{i \in [1, n] : t, i \models \varphi_\alpha, t, i \models \varphi_\tau\}|}{n} \end{cases}$$

For example, Table III shows the probabilities resulting form the evaluation of RCon $(\Diamond \mathsf{b} \wedge \Diamond \mathsf{e}) \boxdot\!\!\rightarrow (\neg \mathsf{c} \vee \Diamond \mathsf{f})$ on trace $\langle \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{f}, \mathsf{c}, \mathsf{e}, \mathsf{c}, \mathsf{h} \rangle$. Probabilities defined as above permit the application, in the context of temporal logic specifications, of measures defined for association rule mining [13], where an

Table II: Probabilistic measures from [13] on association rules in the form *"if A then B"*.

Measure	Formula	Range
Support	$P(AB)$	$[0,1]$
Confidence/Precision	$P(B\|A)$	$[0,1]$
Coverage	$P(A)$	$[0,1]$
Prevalence	$P(B)$	$[0,1]$
Recall	$P(A\|B)$	$[0,1]$
Specificity	$P(\neg B\|\neg A)$	$[0,1]$
Accuracy	$P(AB)+P(\neg A\neg B)$	$[0,1]$
Lift/Interest	$\frac{P(AB)}{P(A)P(B)}$	$[0,+\infty)$
Leverage	$P(B\|A)-P(A)P(B)$	$[-1,1]$
Added Value/ Change of Support	$P(B\|A)-P(B)$	$[-1,1]$
Relative risk	$\frac{P(B\|A)}{P(B\|\neg A)}$	$[0,+\infty)$
Jaccard	$\frac{P(AB)}{P(A)+P(B)-P(AB)}$	$(-\infty,+\infty)$
Certainty factor	$\frac{P(B\|A)-P(B)}{1-P(B)}$	$(-\infty,+\infty)$
Ø-Coefficient (Linear Correlation Coefficient)	$\frac{P(AB)-P(A)P(B)}{\sqrt{P(A)P(B)P(\neg A)P(\neg B)}}$	$(-\infty,+\infty)$
Interestingness Weighting Dependency	$((\frac{P(AB)}{P(A)P(B)})^k-1)\times P(AB)^m$	$[0,+\infty)$
Yule's Q	$\frac{P(AB)P(\neg A\neg B)-P(A\neg B)P(\neg AB)}{P(AB)P(\neg A\neg B)+P(A\neg B)P(\neg AB)}$	$(-\infty,+\infty)$
Yule's Y	$\frac{\sqrt{P(AB)P(\neg A\neg B)}-\sqrt{P(A\neg B)P(\neg AB)}}{\sqrt{P(AB)P(\neg A\neg B)}+\sqrt{P(A\neg B)P(\neg AB)}}$	$(-\infty,+\infty)$
Klosgen	$\sqrt{P(AB)}\times\max(P(B\|A)-P(B),P(A\|B)-P(A))$	$[-1,1]$
Gini Index	$P(A)\times(P(B\|A)^2+P(\neg B\|A)^2)+$ $P(\neg A)\times(P(B\|\neg A)^2+P(\neg B\|\neg A)^2)$ $-P(B)^2-P(\neg B)^2$	$[-2,2]$
Collective Strength	$\frac{P(AB)+P(\neg B\|\neg A)}{P(A)P(B)+P(\neg A)P(\neg B)}\times\frac{1-P(A)P(B)-P(\neg A)P(\neg B)}{1-P(AB)-P(\neg B\|\neg A)}$	$(-\infty,+\infty)$
Laplace Correction	$\frac{N(AB)+1}{N(A)+2}$	$[0.5,1]$
J-Measure	$P(AB)\log\frac{P(B\|A)}{P(B)}+P(A\neg B)\log\frac{P(\neg B\|A)}{P(\neg B)}$	$(-\infty,+\infty)$
Two-Way Support Variation	$P(AB)\log_2\frac{P(AB)}{P(A)P(B)}+P(A\neg B)\log_2\frac{P(A\neg B)}{P(A)P(\neg B)}$ $+P(\neg AB)\log_2\frac{P(\neg AB)}{P(\neg A)P(B)}+P(\neg A\neg B)\log_2\frac{P(\neg A\neg B)}{P(\neg A)P(\neg B)}$	$(-\infty,+\infty)$
Zhang	$\frac{P(AB)-P(A)P(B)}{\max(P(AB)P(\neg B),P(B)P(A\neg B))}$	$(-\infty,+\infty)$
Conviction	$\frac{P(A)P(\neg B)}{P(A\neg B)}$	$[0,+\infty)$
Piatetsky-Shapiro	$P(AB)-P(A)P(B)$	$[-1,1]$
Cosine	$\frac{P(AB)}{\sqrt{P(A)P(B)}}$	$[0,+\infty)$
Loevinger	$1-\frac{P(A)P(\neg B)}{P(A\neg B)}$	$(-\infty,1]$
Information Gain	$\log\frac{P(AB)}{P(A)P(B)}$	$(-\infty,+\infty)$
Sebag-Schoenauer	$\frac{P(AB)}{P(A\neg B)}$	$[0,+\infty)$
Least Contradiction	$\frac{P(AB)-P(A\neg B)}{P(B)}$	$(-\infty,+\infty)$
Odd Multiplier	$\frac{P(AB)P(\neg B)}{P(B)P(A\neg B)}$	$[0,+\infty)$
Example and Counterexample Rate	$1-\frac{P(A\neg B)}{P(AB)}$	$(-\infty,1]$
Odds ratio	$\frac{P(AB)P(\neg A\neg B)}{P(A\neg B)P(\neg BA)}$	$[0,+\infty)$
One-Way Support	$P(B\|A)\log_2\frac{P(AB)}{P(A)P(B)}$	$(-\infty,+\infty)$
Two-Way Support	$P(AB)\log_2\frac{P(AB)}{P(A)P(B)}$	$(-\infty,+\infty)$

Table III: Evaluation (0 is *False* and 1 is *True*), probabilistic interpretation, and statistics computation of a sample of measures for RCon $(\Diamond\!\!\!-\mathsf{b}\wedge\Diamond\mathsf{e})\boxdot\!\!\!\rightarrow(\neg\mathsf{c}\vee\Diamond\mathsf{f})$.

Trace $t_1=$	$\langle$	a,	b,	c,	d,	f,	c,	e,	c,	h	$\rangle$
φ_α: $(\Diamond\!\!\!-\mathsf{b}\wedge\Diamond\mathsf{e})$		0	1	1	1	1	1	1	0	0	
φ_τ: $(\neg\mathsf{c}\vee\Diamond\mathsf{f})$		1	1	1	1	1	0	1	0	1	

$P(\varphi_\alpha)=6/9$	$P(\neg\varphi_\alpha\varphi_\tau)=2/9$	$P(\neg\varphi_\alpha\neg\varphi_\tau)=1/9$
$P(\varphi_\tau)=7/9$	$P(\varphi_\alpha\neg\varphi_\tau)=1/9$	$P(\varphi_\alpha\varphi_\tau)=5/9$

Support: $P(\varphi_\alpha\varphi_\tau)=0.56$ Confidence: $P(\varphi_\tau|\varphi_\alpha)=0.83$

Specificity: $P(\neg\varphi_\tau|\neg\varphi_\alpha)=0.33$ Lift: $P(\varphi_\alpha\varphi_\tau)/(P(\varphi_\alpha)P(\varphi_\tau))=1.07$

	Event log	Support	Confidence	Specificity	Lift
$t_1=$	$\langle$a, b, c, d, f, c, e, c, h$\rangle$	0.56	0.83	0.33	1.07
$t_2=$	$\langle$b, d, a, f, g, d, e, d$\rangle$	0.88	1.00	0.00	1.00
$t_3=$	$\langle$a, c, d, b, c, e, f, c$\rangle$	0.38	1.00	0.20	1.14
$t_4=$	$\langle$b, c, c, e, a$\rangle$	0.4	0.50	0.00	0.83
	Mean	0.55	0.83	0.13	1.01
	Standard deviation	0.23	0.24	0.16	0.13
	Variance	0.05	0.06	0.03	0.02

antecedent A and a consequent B are defined for every rule in the form *"if A then B"*. To that extent, it suffices to map φ_α to A and φ_τ to B, thus having $P(A)$ as $P(\varphi_\alpha)$, $P(B)$ as $P(\varphi_\tau)$, and $P(AB)$ as $P(\varphi_\alpha\cap\varphi_\tau)$. As a result, we can extend any measure defined for association rules to temporal rules as well, including those of Table II. For example, Table III shows few measures computed from the probabilities of RCon $(\Diamond\!\!\!-\mathsf{b}\wedge\Diamond\mathsf{e})\boxdot\!\!\!\rightarrow(\neg\mathsf{c}\vee\Diamond\mathsf{f})$.

C. Measurement system

Given as input an event log L, a set of Reactive Constraints (RCons) R, and a set of probabilistic measures M, our framework returns the measurement of every measure in M for each constraint in R over log L. More precisely, the output can be reported at three different levels of detail:

Event level: distinct evaluation of φ_α and φ_τ for every constraint in R on each event of each trace in L;

Trace level: measurement of every measure in M for every trace in L for every constraint in R;

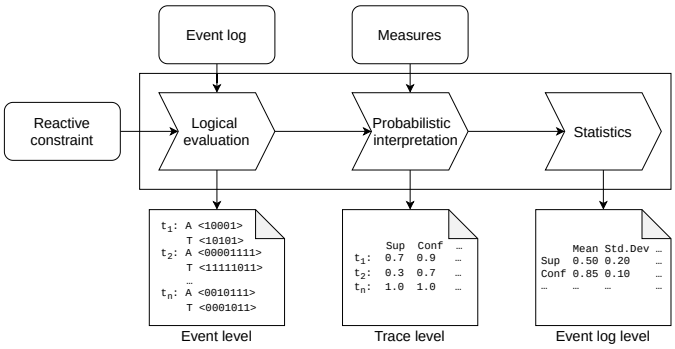


Figure 1: Measurement framework pipeline.

Log level: statistical distribution of every measure in M for every constraint in R.

For example, Table III shows the aggregation of trace-level measures for RCon $(\Diamond\!\!\!-\mathsf{b}\wedge\Diamond\mathsf{e})\boxdot\!\!\!\rightarrow(\neg\mathsf{c}\vee\Diamond\mathsf{f})$ in a log that consists of 4 traces. Being able to perceive the overall status of a constraint is as important as the possibility to analyze its details in single traces. Therefore, we report the entire statistical distribution of a measure across the log to provide a complete information spectrum. Figure 1 depicts the pipeline of the framework from the input to the output. In the first stage, an RCon is evaluated in each trace of the log. Then, the evaluation result is used to compute probabilities and measures of the rule in each trace. In the final stage, the statistical distribution of each measure over the log is drawn.

The design of the RCons, in particular the choice of the activators, is crucial for the evaluation and the computation of the measures. Let us take as an example the constraint RESPONDEDEXISTENCE(a, b) from the repertoire of DECLARE (see Table I). The classical LTL_f formula underlying RESPONDEDEXISTENCE(a, b) for whole-trace evaluations is $\neg\Diamond\mathsf{a}\vee\Diamond\mathsf{b}$ [16]. However, the formulation of the rule as an RCon can lead to different interpretations:

- $\mathsf{a}\boxdot\!\!\!\rightarrow(\Diamond\mathsf{b}\vee\Diamond\!\!\!-\mathsf{b})$, i.e., when a occurs, b is expected to occur somewhere in the trace;
- $(\Diamond\mathsf{a}\vee\Diamond\!\!\!-\mathsf{a})\boxdot\!\!\!\rightarrow(\Diamond\mathsf{b}\vee\Diamond\!\!\!-\mathsf{b})$, i.e., for every event of the trace

Table IV: Measurements of a constraint expressed with different formulations on trace $\langle \mathsf{d}, \mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{a} \rangle$.

RCon formulation	Evaluation	Support $P(\varphi_\alpha \varphi_\tau)$	Confidence $P(\varphi_\tau \mid \varphi_\alpha)$
$\mathsf{a} \boxdot\!\!\rightarrow (\Diamond \mathsf{b} \vee \Diamond\!\!\!-\, \mathsf{b})$	φ_α: $\langle 0,1,0,0,1 \rangle$ φ_τ: $\langle 1,1,1,1,1 \rangle$	$2/5 = 0.4$	$2/2 = 1$
$(\Diamond \mathsf{a} \vee \Diamond\!\!\!-\, \mathsf{a}) \boxdot\!\!\rightarrow (\Diamond \mathsf{b} \vee \Diamond\!\!\!-\, \mathsf{b})$	φ_α: $\langle 1,1,1,1,1 \rangle$ φ_τ: $\langle 1,1,1,1,1 \rangle$	$5/5 = 1$	$5/5 = 1$
$\mathit{True} \boxdot\!\!\rightarrow \neg(\Diamond \mathsf{a} \vee \Diamond\!\!\!-\, \mathsf{a}) \vee (\Diamond \mathsf{b} \vee \Diamond\!\!\!-\, \mathsf{b})$	φ_α: $\langle 1,1,1,1,1 \rangle$ φ_τ: $\langle 1,1,1,1,1 \rangle$	$5/5 = 1$	$5/5 = 1$
$t_{\text{Start}} \boxdot\!\!\rightarrow (\neg \Diamond \mathsf{a} \vee \Diamond \mathsf{b})$	φ_α: $\langle 1,0,0,0,0 \rangle$ φ_τ: $\langle 1,1,1,0,0 \rangle$	$1/5 = 0.2$	$1/1 = 1$

such that a occurs either in the past or in the future, also b should occur somewhere in the trace;

- *True* $\boxdot\!\!\rightarrow \neg(\Diamond \mathsf{a} \vee \Diamond\!\!\!-\, \mathsf{a}) \vee (\Diamond \mathsf{b} \vee \Diamond\!\!\!-\, \mathsf{b})$, i.e., at every event, if a occurs in the trace, also b is expected to occur;
- $t_{\text{Start}} \boxdot\!\!\rightarrow (\neg \Diamond \mathsf{a} \vee \Diamond \mathsf{b})$, i.e., at the beginning of the trace, if a occurs in the trace also b should occur.

All the formulations above are legitimate as they entail that the occurrence of a in the trace demands the occurrence of b. However, the difference in the way the activator is represented turns out to be crucial. The activator, indeed, encodes when the rule is of interest. For example: are we interested in each single occurrence of task a or only in its eventual occurrence in the trace? Do we want the rule to be satisfied in every point of the trace or just at the beginning of the trace? This choice has a clear impact on the measures. Table IV presents the evaluation of a trace with the different formulations seen above and their measurements for the confidence and support measures. While they are all perfectly compliant to the trace (confidence is equal to 1, i.e., each time the activator holds true, also the target holds true), the support varies considerably, i.e., the frequency of $\varphi_\alpha \cap \varphi_\tau$. Notice that this phenomenon comes with neither a good nor with a bad connotation, but stresses the idea that a full control over the formula implies a mindful decision about its design and subsequently on picking the right measures for it.

In summary, we have defined a measurement framework for declarative specifications defined as Reactive Constraints, capable of reporting customizable measures at both trace and event log levels.

V. Implementation and Performance Analysis

We have implemented our measurement framework as a proof-of-concept software prototype on top of an existing declarative process specification discovery tool [7]. The Java source-code can be found at github.com/Oneiroe/Janus. The core software architecture for the verification of RCons is shared with the discovery tool. That is why the new and the old software components are contained in the same repository despite the independence of the two modules (discovery and measurement). All the models used in the following experiments are discovered with [7].

In the remainder of this section, we report on the results of an experimental investigation on the computational performance of our implemented framework. In particular, we assess the performance of the implemented technique against an increase in the data size (i.e., the size of the event log) and the model size (i.e., the number of rules) with synthetic event logs. Finally, we test the performance on a set of real life logs.

Table V: The set of DECLARE rules used in the experiments.

Init(a)	Response(e, f)	ChainResponse(o, p)
End(b)	Precedence(g, h)	ChainPrecedence(q, r)
AtMostOne(c)	AlternatePrecedence(i, l)	RespondedExistence(s, t)
Participation(d)	AlternateResponse(m, n)	

We repeated every experiment 10 times to smooth random factors. The reported results average over the ones of the single repetitions. The machine used for the experiments was equipped with an Intel Core i5-7300U CPU at 2.60GHz, quad-core, 16Gb of RAM and an Ubuntu 18.04.4 LTS operating system.

To test the response of our implemented framework against the input data size, we set up a controlled experiment in which we first generated logs of varying sizes that are compliant with a fixed set of rules, resorting on the simulation engine of MINERful [20]. Thereupon, we computed the measures listed in Table II against all the rules of a larger test model (not fully compliant with the event log). For every run, we recorded the wall-clock time of our prototype.

The starting set of rules stems from the DECLARE repertoire of templates [5] and is provided in Table V. Notice that the set contains all the rule templates seen in Table I and is designed in a way that every constraint insists on different tasks. The test model consists of 649 constraints extracted by the discovery algorithm of Janus (setting the support and confidence threshold parameters to 0.05 and 0.8) from a synthetic event log of 834 963 events, 500 traces and tasks in $[\mathsf{a}, \mathsf{z}]$ that is compliant with the initial model.[1] Given the test model obtained as described above, we performed two tests of 65 iterations each, based on synthetic event logs that comply with the rules of Table V, by: (1) increasing the length of the traces (with a step of 100 events per iteration, keeping the number of traces per event log equal to 500); (2) increasing the number of traces in the event log (with a step of 50 new traces per iteration, keeping the trace lengths between 900 and 1000 events). Figure 2 illustrates the results of both experiments. We observe that the factor actually influencing the wall-clock time is the total amount of events rather than the trace length: indeed, Fig. 2 shows that the recorded timings of both experiments tend to lie on the same line. This experimental result confirms the linear relation between the total number of events in the log and the computational performance illustrated in Section IV-A.

Next, we investigate the response of the framework to an increase in the model size. To do so, we first generated an event log containing 1000 traces with a trace length between 100 and 500 events from the simulation of the rules in Table V. Thereupon, we used the discovery algorithm of Janus to automatically retrieve different test models with varying levels of compliance. To that extent, we made the confidence threshold range from 1.0 (full model compliance), down to 0.0 with a step

[1] Available at github.com/Oneiroe/Janus/blob/master/tests-SJ2T/model.zip

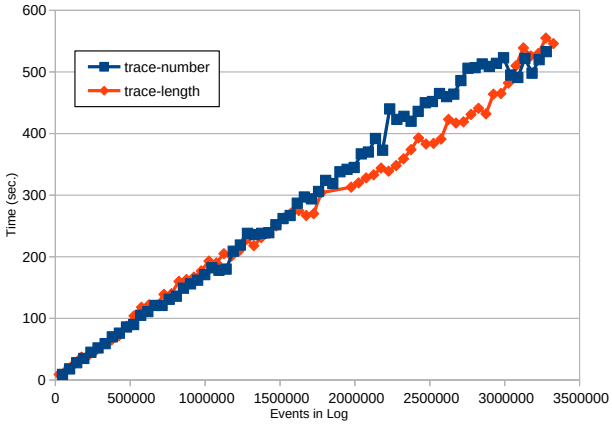


Figure 2: The computation time is linearly dependent on the total number of events in the event log.

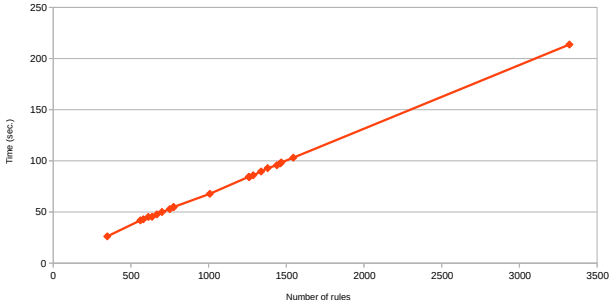


Figure 3: The computation time is linearly dependent on the total number of rules to check.

of 0.05. The rationale is, the lower the confidence threshold, the higher the number of constraints in the test model. Then, we calculated all the measures in Table II for every constraint of each test model. The time taken for the measurements are shown in Fig. 3. Notice that the computation time is linearly dependent on the number of rules to check, thus in line with the theoretical computational cost exposed in Section IV-A.

To test the performance also in real settings, we calculated the measurements on 11 openly available BPICs event logs[2] plus one event log stemming from a partner of a smart-city project in which the authors are involved (labelled as "Smart city" in Table VI). We included the latter event log due to its considerable size: as it can be noticed from the table, it is the one bearing the largest amount of events in this experiment.

For each log, we ran the discovery algorithm of Janus in order to extract a test model to check the event log against. We tuned the parameters of the discovery algorithm to obtain a set of rules that are highly compliant (confidence threshold of 0.8), even if not frequent (support threshold of 0.05). Table VI illustrates the results. For each log we report, along with the number of traces, tasks and events contained in the log and the number of rules in the test model, the total time from the launch to the termination of the software ("Total"), the time to evaluate the rules on every event ("Checks"), and the time to compute the measures aggregated by trace and log ("Measures"). We remark that the wall-clock time remains within acceptable ranges as the slowest run takes around 1.5 minutes to check about 500 constraints.

[2] https://data.4tu.nl/

Table VI: Performance records on real-life datasets.

Event log	Traces	Tasks	Events	Rules	Total [sec]	Check [msec]	Measures [msec]
BPIC12	13087	36	262200	519	94.7	20 589.40	70 414.90
BPIC13_cp	1487	7	6660	20	1.3	129.3	274.9
BPIC13_i	7554	13	65533	14	2.6	389.5	666.5
BPIC14_f	41353	9	369485	51	20.6	3871.70	13 477.10
BPIC15_1f	902	70	21656	3856	37.9	13 197.70	22 796.00
BPIC15_2f	681	82	24678	5889	56	19 199.60	35 130.60
BPIC15_3f	1369	62	43786	4098	72.3	24 449.00	45 503.70
BPIC15_4f	860	65	29403	4690	55	18 605.10	34 459.80
BPIC15_5f	975	74	30030	5164	62.9	21 975.00	39 039.20
RTFMP	150370	11	561470	49	62	12 666.30	43 145.40
SEPSIS	1050	16	15214	260	3.1	710.70	1590.70
Smart city	4347	20	692333	292	67	22 226.60	39 295.50

VI. Evaluation

In this section, we report on experiments that show interesting implications of having a vast availability of measures with customization options. We argue that having different measures yields a more precise characterization of the behavior at hand. Indeed, different measures respond differently to different stimuli in the data. To support this claim, we study the variation in the measures at the injection of specific noise types in the event log that directly or indirectly affect a constraint, thereby assessing their sensitivity or resilience to changes. In fact, while an observable reaction of measures could be desirable when, e.g., noise reveals a change in the process execution, resilience could be preferable, for example, when noise stems from a mere interference or technical issue affecting the information system that records the log. Ultimately, we observe how different measures sense different aspects of a constraint, thus confirming the need of using a multitude of measures and the importance of a proper selection thereof to conduct an in-depth behavioral analysis.

We conducted the experiment as follows. We took as reference model the set of rules in Table V and the synthetic event log that complies with it.[1] Notice that the rules are designed not to interfere with one another as each of them insists on different tasks. In this way, it is possible to observe the response of measures at varying noise levels targeting one constraint at a time, thus diminishing the effect of cross-interference. Thereupon, we injected noise in the event log, resorting on the technique described in [21], and calculated all the measures in Table II for the reference model. In particular, we made use of the following types of noise:

Events insertion: spurious events are included in the traces (mimicking, e.g., double records, alien events, etc.);

Events deletion: events are expunged from the log (mimicking, e.g., missing records, uncommitted transactions, etc.);

White noise: events are randomly inserted and deleted.

Addressing one rule at a time, we studied (1) the direct effect of noise on that constraint by altering the occurrences of its

activator and target via insertions and deletions, and (2) the indirect effect, by altering the occurrences of the other tasks in the log with white noise. We made the noise spread all over the log according to a controlled probability variable. For instance, setting the noise injection as the deletion of occurrences of task a with a probability of 20% results in the removal of 20% of the occurrences of task a from the log, picked at random.

More specifically, for every rule in the set of Table V, we ran a separate experiment for (i) event insertion noise affecting the activator or (ii) the target, (iii) event deletion noise affecting the activator or (iv) the target, (v) white noise affecting neither the activator nor the target. For each of the combinations above, we let the error-injection probability range from 0 to 100 % with a step of 10 %. Because of the random factor, we repeated each experiment 10 times and recorded the average results.

Figure 4 shows the results of such experiments on constraint RESPONSE(e, f). Every line corresponds to a measure. The RESPONSE constraint imposes that the target occurs eventually after each occurrence of the activator. The plots reveal through the starting point and the steepness of the slopes in the curves whether, and to what extent, the corresponding measures consider the frequency of the events satisfying the activator or the target in the traces. As it can be seen, the measures shows different trends for each stimulus.

At large, the RESPONSE constraint appears to be particularly sensitive to the deletions of the target and influenced by both the deletions and insertions of the activator, while mostly insensitive to spurious insertions of the target and white noise. More specifically, we can derive the following observations.

- The deletion of events satisfying the target leads to more violations of the rule (higher $P(\varphi_\alpha \neg\varphi_\tau)$), thus the negative effect is reflected in the drop of many measures based on $P(\varphi_\alpha\varphi_\tau)$ (e.g., confidence, lift, certainty factor). In contrast, the curves of specificity and accuracy grow because of their definition bound to $P(\neg\varphi_\alpha\neg\varphi_\tau)$.
- The deletion of events satisfying the activator, instead, does not bring more rule violations, but only less satisfactions (lower $P(\varphi_\alpha\varphi_\tau)$). Therefore, measures such as relative risk, lift, Zhang, or leverage, decrease. Measures focusing on the target (e.g., prevalence) are basically unaffected.
- The insertion of more events satisfying the target (higher $P(\varphi_\tau)$) does not influence the frequency with which the activator is satisfied. Most of the measures are stable, with a slightly decreasing trend for, e.g., lift or accuracy.
- The insertion of more events satisfying the activator is less characterizing, as the new elements may bring both new rule satisfactions (higher $P(\varphi_\alpha\varphi_\tau)$) and violations (higher $P(\varphi_\alpha\neg\varphi_\tau)$). That is why many measures show an increasing trend (e.g., cosine or Laplace), while others oscillate around the initial value (e.g., confidence). Even the most visible downward trends (see certainty factor, Zhang and Ylue's Y and Q) do not show a totally smooth trend.
- Lastly, the constraint is mostly stable against random alterations that affect neither the activator nor the target. The satisfactions and violations remain constant, whilst the only increase is in $P(\neg\varphi_\alpha\varphi_\tau)$ and $P(\neg\varphi_\alpha\neg\varphi_\tau)$. The slight fluctuations of the measures are due to the variation in the number of events in the traces.

Because of space limits, we cannot illustrate the results for the other rules of Table V. The interested reader can find them at github.com/Oneiroe/Janus/blob/master/tests-SJ2T/NOISE-INJECTION-PLOTS.zip. Overall, generalizing from this specific case, we observe the following: (1) The reaction of the measures to noise depends on the type of stimulus and on whether it affects the target or the activator of the rule. (2) Not all measures sense certain alterations. If a measure is mostly stable with and without an error, it means that it cannot sense that particular stimulus. (3) Some measures have similar trends, but different magnitude. This means that there are "classes" of measures that focus on the same aspects of a rule. (4) The steepness of the curves with which the measure evolves indicates how much the measure is resilient to the change in presence of noise. In particular, it shows the range of tolerance before the error becomes too large to recognize the specific constraint behavior. Notice that sometimes it is desirable to sense if the fundamental characteristics of a log are still visible despite the deviations (e.g., to implement discovery algorithms that are robust to noise).

VII. CONCLUSION

In this paper, we presented a comprehensive measurement framework for declarative specifications modeled as Reactive Constraints. Given an event log and a set of custom probabilistic measures, the framework accepts in input any RCon and returns as output the evaluation of the rule for each event of the log, the computed measures for all the traces, and their statistics over the entire event log. The framework goes beyond the current state of the art as it is not limited to a specific set of measures or rules. The experiments conducted reveal the possibility to characterize the behavior of a given constraint through the combination of different measures, which sense differently the behavior recorded in the log.

Future work. Different possibilities are now open upon the foundations of the measurement framework. It is possible to exploit the possibility to characterize a phenomenon by studying the evolution of different measures for, e.g., dynamic recognition of exceptions in process monitoring [22] or the identification of process drifts [23].

As different measures react differently to different stimuli for different types of rules, a method to select and combine the most appropriate measures depending on the context turns out to be key. To this extent, future research could resort to existing techniques like [24] or develop novel multi-measure heuristics. Also, the measures can be integrated for the assessment of multi-constraint specifications as a whole as in [1], [11].

Many specifications can be considered equivalently valid for static measurements, yet their evolution can show which of them is the most error-tolerant. In this sense, the framework can support the identification of resilient specifications.

While the analysis of multiple measures at once may be overwhelming for a human, machine learning techniques, dealing naturally with multidimensional data, could benefit

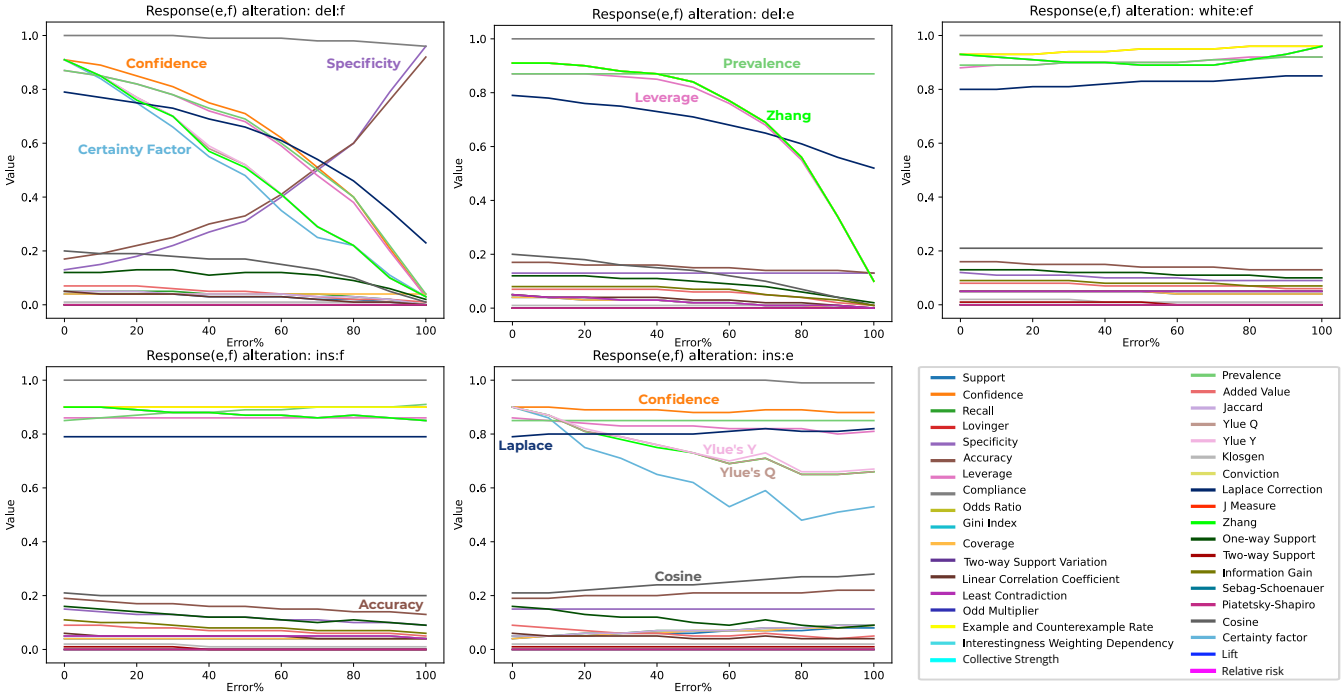


Figure 4: Effect of error injection on constraint RESPONSE(e, f) for all measures.

from the availability of the great amount of information returned by the proposed framework. Therefore, it seems to be also promisingly exploitable for feature selection tasks in sequence classification [25].

Acknowledgment. The work of C. Di Ciccio was partly supported by MIUR under grant "Dipartimenti di eccellenza 2018-2022" of the Department of Computer Science at Sapienza University of Rome.

REFERENCES

[1] M. de Leoni, F. M. Maggi, and W. M. P. van der Aalst, "An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data," *Inf. Syst.*, vol. 47, pp. 258–277, 2015.

[2] 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," *Inf. Syst.*, vol. 54, pp. 209–234, 2015.

[3] F. M. Maggi, R. P. J. C. Bose, and W. M. P. van der Aalst, "Efficient discovery of understandable declarative process models from event logs," in *CAiSE*, 2012, pp. 270–285.

[4] C. Di Ciccio and M. Mecella, "On the discovery of declarative control flows for artful processes," *ACM Trans. Management Inf. Syst.*, vol. 5, no. 4, pp. 24:1–24:37, 2015.

[5] W. M. P. van der Aalst and M. Pesic, "DecSerFlow: Towards a truly declarative service flow language," in *WS-FM*, 2006, pp. 1–23.

[6] T. B. Le and D. Lo, "Beyond support and confidence: Exploring interestingness measures for rule-based specification mining," in *SANER*, 2015, pp. 331–340.

[7] A. Cecconi, C. Di Ciccio, G. De Giacomo, and J. Mendling, "Interestingness of traces in declarative process mining: The Janus LTLp_f approach," in *BPM*, 2018, pp. 121–138.

[8] S. Debois, T. T. Hildebrandt, P. H. Laursen, and K. R. Ulrik, "Declarative process mining for DCR graphs," in *SAC*, 2017, pp. 759–764.

[9] W. Hämäläinen and G. I. Webb, "A tutorial on statistically sound pattern discovery," *Data Min. Knowl. Discov.*, vol. 33, no. 2, pp. 325–377, 2019.

[10] 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," *ACM Trans. Softw. Eng. Methodol.*, vol. 29, no. 3, Jun. 2020.

[11] A. Burattin, F. M. Maggi, W. M. P. van der Aalst, and A. Sperduti, "Techniques for a posteriori analysis of declarative processes," in *EDOC*, 2012, pp. 41–50.

[12] J. Yang, D. Evans, D. Bhardwaj, T. Bhat, and M. Das, "Perracotta: mining temporal API rules from imperfect traces," in *ICSE*, 2006, pp. 282–291.

[13] L. Geng and H. J. Hamilton, "Interestingness measures for data mining: A survey," *ACM Comput. Surv.*, vol. 38, no. 3, p. 9, 2006.

[14] M. Gabel and Z. Su, "Online inference and enforcement of temporal properties," in *ICSE*, 2010, pp. 15–24.

[15] C. Lemieux, D. Park, and I. Beschastnikh, "General LTL specification mining (T)," in *ASE*, 2015, pp. 81–92.

[16] G. De Giacomo and M. Y. Vardi, "Linear temporal logic and linear dynamic logic on finite traces," in *IJCAI*, 2013, pp. 854–860.

[17] M. Pesic, D. Bosnacki, and W. M. P. van der Aalst, "Enacting declarative languages using LTL: avoiding errors and improving performance," in *SPIN*, ser. LNCS, vol. 6349. Springer, 2010, pp. 146–161.

[18] C. Baier, J.-P. Katoen, and K. Guldstrand Larsen, *Principles of Model Checking*. The MIT Press, 2008.

[19] M. Dumas, M. La Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management, Second Edition*. Springer, 2018.

[20] C. Di Ciccio, M. L. Bernardi, M. Cimitile, and F. M. Maggi, "Generating event logs through the simulation of declare models," in *EOMAS*, 2015, pp. 20–36.

[21] C. Di Ciccio, M. Mecella, and J. Mendling, "The effect of noise on mined declarative constraints," in *SIMPDA*, 2013, pp. 1–24.

[22] F. M. Maggi, M. Montali, M. Westergaard, and W. M. P. van der Aalst, "Monitoring business constraints with linear temporal logic: An approach based on colored automata," in *BPM*, 2011, pp. 132–147.

[23] A. Yeshchenko, C. D. Ciccio, J. Mendling, and A. Polyvyanyy, "Comprehensive process drift detection with visual analytics," in *ER*, 2019, pp. 119–135.

[24] Z. Cao, Y. Tian, T. B. Le, and D. Lo, "Rule-based specification mining leveraging learning to rank," *Autom. Softw. Eng.*, vol. 25, no. 3, pp. 501–530, 2018.

[25] Z. Xing, J. Pei, and E. J. Keogh, "A brief survey on sequence classification," *SIGKDD Explorations*, vol. 12, no. 1, pp. 40–48, 2010.

Rule Mining with RuM

Anti Alman
University of Tartu
anti.alman@ut.ee

Claudio Di Ciccio
Sapienza University of Rome
diciccio@di.uniroma1.it

Dominik Haas
WU Vienna
dominik.haas@s.wu.ac.at

Fabrizio Maria Maggi
Free University of Bolzano
maggi@inf.unibz.it

Alexander Nolte
University of Tartu
alexander.nolte@ut.ee

***Abstract*—Declarative process modeling languages are especially suitable to model loosely-structured, unpredictable business processes. One of the most prominent of these languages is Declare. The Declare language can be used for all process mining branches and a plethora of techniques have been implemented to support process mining with Declare. However, using these techniques can become cumbersome in practical situations where different techniques need to be combined for analysis. In addition, the use of Declare constraints in practice is often hampered by the difficulty of modeling them: the formal expression of Declare is difficult to understand for users without a background in temporal logics, whereas its graphical notation has been shown to be unintuitive. In this paper, we present RuM, a novel application for rule mining that addresses the above-mentioned issues by integrating multiple Declare-based process mining methods into a single unified application. The process mining techniques provided in RuM strongly rely on the use of Declare models expressed in natural language, which has the potential of mitigating the barriers of the language bias. The application has been evaluated by conducting a qualitative user evaluation with eight process analysts.**

***Index Terms*—Rule Mining, Process Analytics Tool, Declarative Process Models, Natural Language Processing**

I. Introduction

Business Process Management (BPM) has become an integral part of how companies organize their workflows starting from the higher levels of management as recommended by ISO 9001 Quality Management Principles (especially principles 4 and 6) [1] to modeling and optimizing lower level processes through the use of various process mining techniques [2]. Process mining is the part of BPM that is focused on the analysis of business processes based on process execution logs (event logs). Process mining techniques can be based on two different approaches for representing process models that are used as their input and/or output: procedural process models or declarative process models.

Procedural process models aim at describing end-to-end processes and allow only for the process behavior that is explicitly specified in the model [3]. However, modeling step by step the entire control-flow of a business process can be cumbersome in some cases. For example, if the process is loosely-structured and has a high number of different paths and exceptions the model could become quickly unreadable. In these cases, it may be a better choice to use declarative process models that model the process as a set of rules that the process should follow. In this way, everything that is not constrained is allowed and several execution paths can be represented in a compact model.

In contrast with the multiple process mining applications available for working with procedural models, there are currently no similar applications for working with declarative models [4]. This lack of a comprehensive toolset can be considered as one of the main contributing factors of the relatively low adoption rate of Declare in the industry and has been named as one of the open research challenges in declarative process mining [5, RC7]. In addition to this, dealing with declarative process models is known to be difficult, especially for domain experts that generally lack expertise in temporal logics and, in most of the cases, find the graphical notation of Declare constraints unintuitive [6].

In this paper, we present RuM,[1] a novel process mining application that addresses these research challenges by integrating multiple Declare-based process mining techniques into a single unified application and by largely making use of natural language to express Declare constraints. With this tool, the user is not required to have any experience in temporal logics nor to be familiar with the graphical notation of Declare constraints, but can handle temporal properties using natural language statements. RuM implements process mining techniques based on MP-Declare [7] the multi-perspective extension of Declare supporting data constraints together with control-flow constraints. RuM also provides a model editor that is fully MP-Declare compliant and equipped with a chatbot that supports inexpert users in defining Declare constraints using natural language expressions.

To assess the feasibility of RuM, we conducted a qualitative user evaluation. Our aim was to (1) gain insights into how users from different backgrounds perceived the application and (2) identify means for improving it. In general, the application was well received and it was recognized to be timely and highly needed by all participants.

The remainder of this paper is structured as follows. Section II gives a short overview of the Declare language. Section III discusses the main design goals of RuM. Section IV gives an overview of the functionalities of RuM and lists the process mining techniques available in the application. Section V describes the user evaluation methodology and provides an overview of the evaluation results. Finally, Section VI concludes the paper and spells out directions for future work.

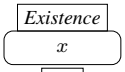
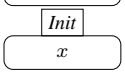
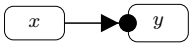
II. Basics of Declare

Declare is a modeling language that uses a constraint-based declarative approach to model loosely-structured processes

[1] https://rulemining.org

978-1-7281-9832-3/20/$31.00 ©2020 IEEE
DOI 10.1109/ICPM49681.2020.00027

TABLE I
SOME DECLARE TEMPLATES

Template	Explanation	Notation
Unary constraints		
EXISTENCE(x)	Activity x occurs at least once per trace	
INIT(x)	Activity x occurs at the beginning of every trace	
Binary constraints		
RESPONSE(x,y)	If x occurs, then y must occur eventually after x	
CHAINRESPONSE(x,y)	If x occurs, then y must occur immediately after x	
PRECEDENCE(x,y)	y occurs only if preceded by x	

through behavioral constraints [8]. The language is grounded in linear temporal logic over finite traces [8], [9].

Declare is based on templates, which are parameterized temporal logic patterns and come endowed with a graphical notation to ease the depiction of process maps. To define process models using Declare, no knowledge of the underlying formal logic is required since Declare templates can also be expressed as natural language sentences. Declare constraints are concrete instantiations of templates obtained by replacing template parameters with real activities.

Table I shows some templates that will be used throughout the paper. Unary templates exert conditions on the occurrence of single activities. For example, EXISTENCE(x) requires activity x to occur in a trace. Binary templates predicate over pairs of activities. For instance, RESPONSE(x,y) imposes that if x occurs, then y must eventually occur afterward. Binary templates partition their parameters into an activation and a target. The activation triggers the constraint, like the antecedent of a logical implication (e.g., x for RESPONSE(x,y)). The target is subject to the restriction exerted by the occurrence of the activation, like the consequent of a logical implication (e.g., y for RESPONSE(x,y)). Declare also includes *negative* versions of the binary templates: e.g., NOTRESPONSE(x,y) states that if x occurs, no y can occur afterward.

A Declare model consists of a set of constraints that hold under logical conjunction. The behavioral semantics of every constraint can be represented by means of a finite-state automaton. A Declare model can thus be represented as a finite-state automaton stemming from the synchronous product of the automata of the single constraints [10].

More recently, the addition of conditions that predicate not only on activities and their control-flow, but also on data attributes and timestamps led to the definition of an extension

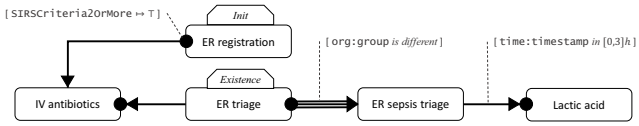


Fig. 1. An MP-Declare map

of standard Declare, namely MP-Declare [7]. Depending on the parameters on which data conditions insist, they are classified as *activation* conditions, *target* conditions or *correlation* conditions (the latter being exerted on both parameters). On the other hand, *time* conditions express constraints over the time distance between the activation and the target of a constraint.

Figure 1 depicts the graphical representation of an MP-Declare model (a map). The model is inspired by the analysis of a real-world event log[2] by Mannhardt et al. [11], [12] and refers to the healthcare process of handling patients affected by sepsis. The process begins with the emergency room (ER) registration. After that, if the activation condition on the Systemic Inflammatory Response Syndrome (SIRS) criteria attribute holds true, then intravenous (IV) antibiotics have to be administered. The IV antibiotics activity can only occur if ER triage was executed beforehand. Immediately after ER triage, the ER sepsis triage follows, but with the correlation condition that the actors carrying out the two activities differ (correlation condition on the `org:group` attribute). The analysis of lactic acid presence requires the ER sepsis triage to be run beforehand within 0 and 3 hours (time condition).

III. DESIGN GOALS

In this section, we give a short overview of the main design goals of RuM. All the design goals are based on the combination of (1) the principle of "know your users" [13] and (2) the intended purpose of RuM [5, RC7]. We identified the target audience to be researchers and industry experts who may have varying levels of experience with Declare or declarative models in general. The main objective of RuM is to provide a comprehensive toolset for working with process mining techniques based on declarative models. Based on this, we identified the following 4 design goals.

First, the UI must have a low threshold for use and avoid an initial steep learning curve [14] especially since the target audience includes users that are not necessarily familiar with Declare. To achieve this, we decided to follow a minimalist visual design [15] and to divide the UI into different higher-level views based on the different tasks a user might want to carry on such as discovering a process model or checking the conformance of an event log.

Second, multiple different methods must be supported for each functionality (based on [5, RC7]). To achieve this, we selected and integrated multiple well-known Declare-based process mining techniques into RuM (Section IV). Additionally, we designed RuM in such a way that multiple process

[2] http://dx.doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460

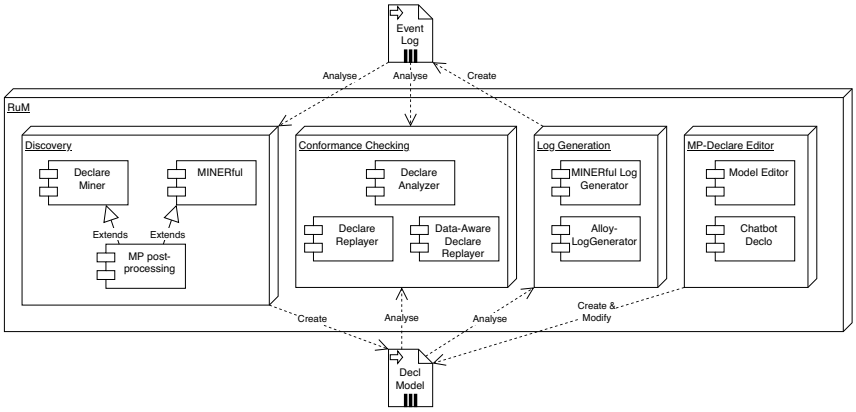


Fig. 2. Main functionalities of RuM

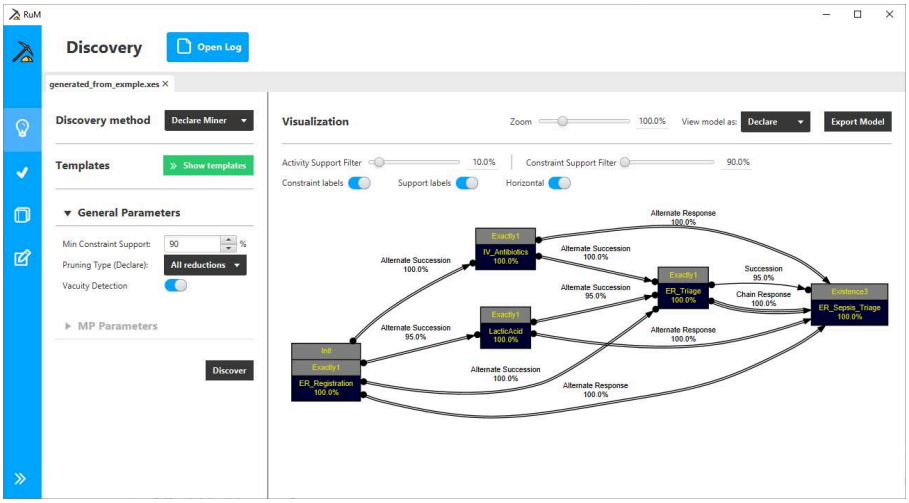


Fig. 3. Discovery UI

mining tasks can be started simultaneously. The user is thus free to navigate the rest of the application while a process mining task is ongoing.

Third, the functionalities in RuM should be easily reachable (principle of reachability [13]). To achieve this, we placed the input parameters of process mining methods into the same views where results are displayed or at most one click away via the use of slide-in panels. Additionally, we designed the UI of RuM in such a way that the results of two different process mining tasks are always at most two clicks away from each other (via the use of a side menu for navigating the main functionalities and a row of tabs for navigating the results).

Fourth, the parameters of different methods in the same functionality must be similar where possible (principle of consistency [13]). This is achieved by using the same input fields for all parameters that are common for different methods. If a parameter is specific to a selected method (for example process discovery techniques use different pruning approaches) then this parameter is explicitly labeled as method specific. The same counts for the results of different methods that must also be comparable. This was achieved by creating result views that have identical structure regardless of which method is used.

IV. Functional Overview

RuM is the first software platform natively designed to analyze processes using a rule-based approach. To this end, we have integrated and improved existing prototypes, but also created completely new features that enhance the user experience during the process analysis. To cater for the interoperability of the tool, we resort on existing standards for input and output files, namely XES [16] for the event logs and *decl* [17] for the models. The diagram in Fig. 2 illustrates the software architecture of RuM with the components implementing its main functionalities: process discovery, conformance checking, log generation, and model editor. A demonstration video of these functionalities is available at: https://youtu.be/_6IMwR_SwaQ. In the following, we describe them in detail.

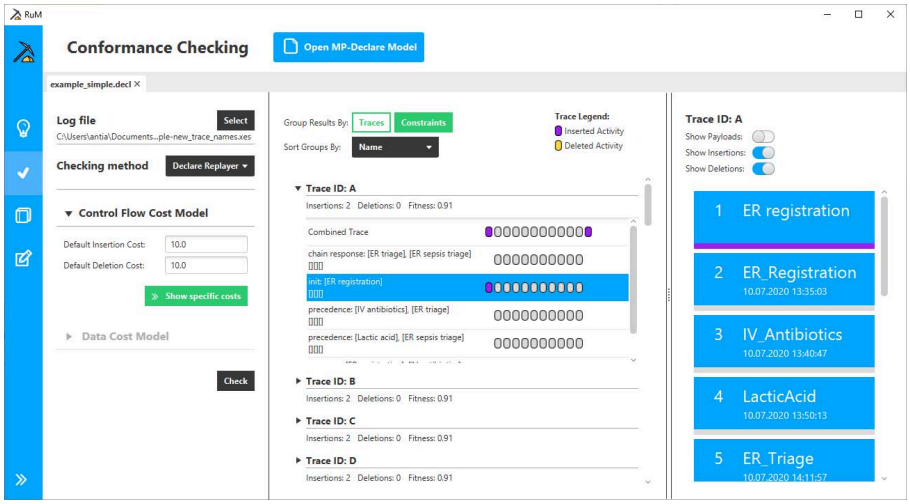


Fig. 4. Conformance Checking UI

A. Discovery

Four methods are available for process discovery: Declare Miner [18], MINERful [19], MP-Declare Miner [20] and MP-MINERful. The MP variants add to the base mining algorithms a post processing step for discovering data conditions.[3]

The input parameters and the results are presented in the same view, as illustrated in Fig. 3. The discovery results can be explored by using three complementary views: through a process map (Declare view), a textual description (textual view), or as a procedural model (automaton view). In the remainder of this section, we describe them more in detail. We remark that those views support filtering based on activity support and constraint support thus providing the possibility for users to show/hide outlier behaviors.

1) Declare View: The Declare view represents the discovered model using the standard graphical notation for Declare constraints. Each activity in the model is represented as a single rectangle containing the activity name and the activity support (i.e., the percentage of log traces in which the activity occurs). Notice that the background of the activity rectangle is colored based on its support for immediacy of information conveyance. For constraints, it is possible to show or hide both the constraint name and the constraint support.

2) Textual View: The textual view is a model representation meant for users who are less familiar with the graphical syntax of Declare. The aim of the textual view is to describe the model using natural language sentences that are easy to understand without any prior knowledge of Declare.

3) Automaton View: The automaton view displays the discovered process model as a finite-state machine. This view is meant for users who are familiar with the formal semantics of Declare.

[3]Notice that all the "data-aware" versions of the techniques provided in RuM support a richer language at the expense of lower efficiency.

B. Conformance Checking

There are three methods available for conformance checking: Declare Analyzer [7], Declare Replayer [21] and Data-Aware Declare Replayer.[4] The Declare Analyzer takes as input a model and an event log, and returns activations, violations, and fulfillments in the log of each constraint in the model. The Declare Replayer and the Data-Aware Declare Replayer report trace alignments. The Data-Aware Declare Replayer can also account for the data perspective. Both input parameters and results are presented in the same view (Fig. 4).

The conformance checking results are presented in groups. Each group represents the results for a specific trace or a specific constraint. Each group can be expanded to see the result details of that group. This allows users to explore the results at a high level of detail while also being relatively compact in terms of user interface. Notice that, if the result is a trace alignment, then it is also possible to show or hide both the activities that are inserted into the trace or removed from the trace as a result of the alignment.

C. MP-Declare Editor

In order to provide a comprehensive toolset to work with Declare process models, RuM contains a model editor that supports not only standard Declare but also MP-Declare. The MP-Declare editor uses the *decl* file format to import and export the models. All aspects of the format are supported: activity definitions, attribute definitions, activity-attribute bindings and constraints with all the allowed data and time conditions.

The MP-Declare editor is presented in a single view (Fig. 5). Two slide-in panels are used, the first one for editing the activities and the second one for editing the attributes. Editing the constraints is done in a single table where each row corresponds to a single constraint. The entire model is also

[4]https://github.com/Clyvv/DataAwareDeclareReplayer

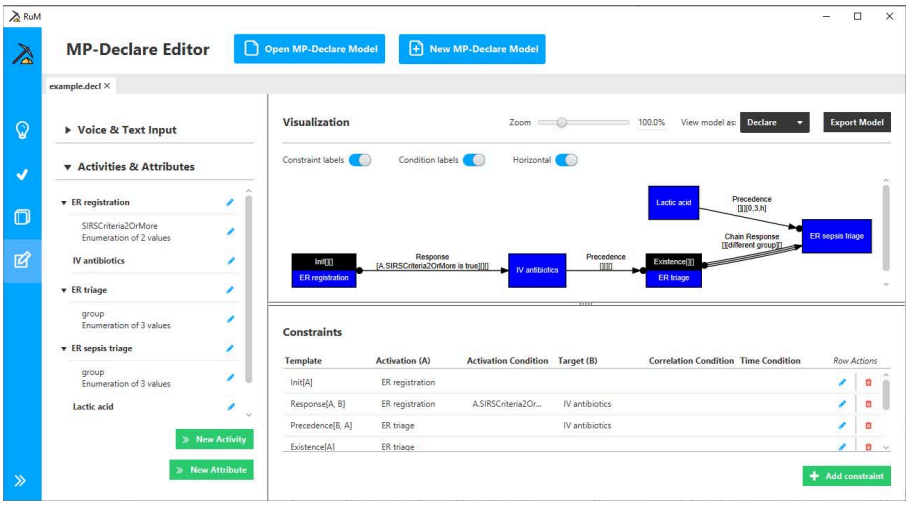


Fig. 5. MP-Declare Editor UI

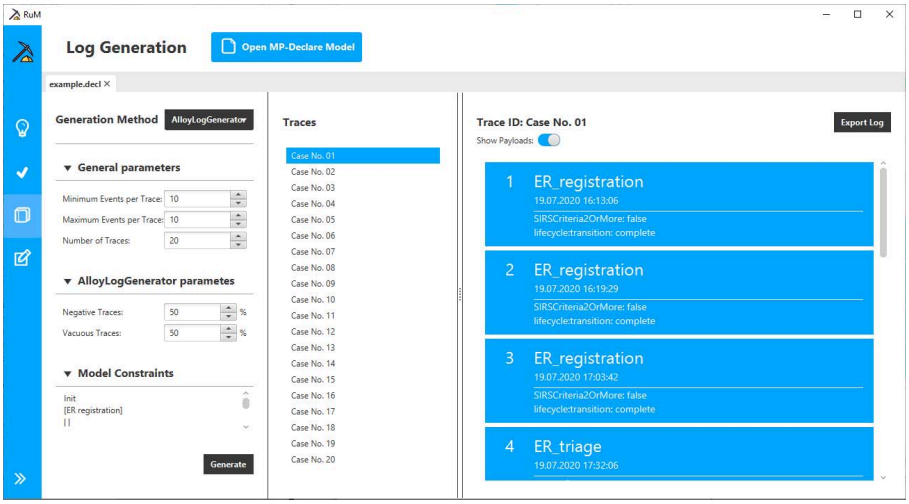


Fig. 6. Log Generation UI

represented visually in the same view and the visualization is updated on-the-fly as the user is editing the model. The used visualization devices are the same as those of the discovery panel, i.e., the model can be visualized using the standard Declare graphical notation, as text or in the form of an automaton.

Finally, in the editor, users can also add constraints and data conditions using natural language sentences. The sentences can be provided both by voice and text. This functionality is implemented as a simple chatbot named Declo [22].

D. Log Generation

There are two log generation methods available in RuM: AlloyLogGenerator [17] and MINERful Log Generator [10]. The main difference between these methods from the user's standpoint is that the AlloyLogGenerator can also account for the definition of activation and correlation conditions in the input process model. Both the input parameters and the generated log are presented in a single view as illustrated in Fig. 6. Among other options, the user can specify the percentage of traces that trivially satisfy the constraints in the input model (i.e., traces that comply with the constraints because their activation never occurs) and the percentage of negative traces (i.e., traces that violate at least one of the constraints in the input model). The generated log can be exported in XES format.

V. User Evaluation

To assess the feasibility of RuM, we conducted a qualitative user evaluation. Our aim was to (1) gain insights into how users from different backgrounds perceived the application and (2) identify means for improving it. In the following, we will write "(obs.)" to mark findings that are based on the observation of the participants while using the application.

A. Study Setup

We selected eight process analysts as participants for our study. Four participants had little to no knowledge of Declare (B1 to B4), but worked in the BPM field (here called BPM experts), while the other four identified themselves as Declare experts (D1 to D4). We chose this differentiation to particularly study the potential differences related to their demands and perception about RuM. Prior to the study the participants were given access to a scenario,[5] some input files,[5] and the application itself. A common scenario was used for all participants in order to ensure the comparability of our findings.

The study was conducted via Skype by a team consisting of a *facilitator* and an *observer*, with the facilitator guiding the participant and the observer serving in a supporting role. It started with the facilitator introducing the study procedure and the application to the participants who were then asked to start RuM on their computer, share their screen and carry out the tasks based on the scenario provided earlier. Participants were encouraged to think aloud, to ask questions and to point out interesting aspects of the application during the test. Every test was video-recorded. After finishing the tasks, the facilitator conducted a short post-interview asking questions about the application in general and about the tasks where the participant appeared to have had difficulties.

Each study lasted between 45 and 60 minutes. After the end of the study, the participants received a link to a short post-survey[5] including the System Usability Scale (SUS) [23] and scales covering satisfaction, expectation confirmation, future use intentions, and usefulness [24]. To analyze the collected data, we focused on the video recordings, observations and follow-up interviews, using the post-surveys as an additional qualitative data point. We followed the affinity-diagramming method [25], which yielded 540 items that were divided into 10 main clusters and 111 sub-clusters in total.

B. Study Scenario

For the study, we developed a scenario that involves common activities of a process analyst such as the discovery of a model from an existing event log and its validation and modification. We used the Sepsis Cases event log that is based on real-life treatment cases[2] as a basis and split it into a training and a test set. The participants were then first asked to use RuM to discover an initial model using the training set (Section V-D1). Afterward, they were instructed to check if the discovered model conforms to the test set using different conformance checking methods (Section V-D2). The participants were then asked to modify the discovered model based on the results of the conformance checking and some additional domain information (Section V-D3). Finally, the participants were asked to use the modified model to generate a new log (Section V-D4).

[5]The evaluation material is available at: https://git.io/JJIp4 (scenario); https://git.io/JJIpu (input files); https://git.io/JJLvB (post-survey).

TABLE II
SURVEY RESULTS REPRESENTED AS AVERAGES FOR BOTH GROUPS AND OVERALL. THE SUS SCORE RANGES BETWEEN 0 AND 100, WHILE THE OTHER SCALES RANGE BETWEEN 1 AND 5.

	Overall	Declare experts	BPM experts
SUS	81.875	78.75	85
Satisfaction	4.5	4.5	4.5
Expectation	4.56	4.33	4.78
Future intentions	4.167	3.833	4.5
Usefulness	4.3125	4.25	4.375

C. General Findings

Both groups of users found the UI of RuM to be usable as evident by statements such as *"nice interface"* (B1), *"I think it's a really nice tool, I really like it"* (D1), *"I think it's really cool"* (D4), *"I was impressed"* (D4). B3 also pointed out that everything in the user interface was understandable *"after clicking around for a few minutes"* (B3). These statements are underpinned by an average SUS score of 81.875 (a SUS score of 69.69 is considered average, while a score above 80 is considered to be good or excellent [26]). It was also pointed out that there is a need for an application like RuM *"I think it's very promising and also very much needed"* (B2), *"I think in the process mining community there was really need to freshen up Declare"* (D1).

However, there was a significant difference between BPM and Declare experts as evident by the post-survey results (Table II). RuM was rated higher by BPM experts on all scales except satisfaction, which was rated as 4.5 by both groups. The largest differences between BPM and Declare experts are in the SUS score (85 to 78.75) and future use intentions (4.5 to 3.833). This discrepancy can point towards Declare experts being already used to existing tools for Declare-based process mining and potentially being less sensitive to the improvements in the ease of use of Declare constraints.

D. Task-Specific Findings

A finding that was orthogonal to all tasks was the fact that, when it was needed to use the result of a section as input in another section, we deliberately did not mention exporting the results. While the process of exporting itself was not an issue, some participants (B2, D4) also expected the application to have quick ways to move files from one section to another with D4 stating *"it's a bit strange that you have to export the model and then reopen it, the same one, in another tab"* (D4). Other findings (described in the following) were task specific.

1) Discovery: The ordering of templates in the template selection panel of the discovery section is based on template categories: unary templates, positive binary templates and negative binary templates. All BPM experts had difficulties in finding the correct templates from this panel, while the same was observed with only one Declare expert (*"ok, so they are not alphabetically sorted"*, D1). For example, B1 scrolled the template list from end to end multiple times before finding all the templates listed in the scenario (obs.). During the post-interview B4 suggested to add *"brief headings"* (B4) because

that would make it *"easier to quickly categorize it"* (B4). It was also mentioned that the large number of templates might be difficult to use (*"we have a lot of templates available which is maybe not straightforward"*, B3).

The second noticeable difference between the two groups was that three out of four BPM experts started working with the initial model that is discovered automatically by the tool with the default parameters (obs.), while our scenario explicitly asked them to use a different set of parameters. None of the Declare experts had the same issue (obs.) with only one mentioning the automatic discovery *"it's interesting it immediately starts to discover something before I could set the parameters"* (D3). The problem was that, even if they changed the parameters, most of the BPM experts did not restart the discovery with the new parameters. This confusion might be related to filters that (differently from parameters) are applied on-the-fly to the discovered model (*"I did not do that because the map responded to some of the things I changed, for instance when I adapted the sliders"*, B2).

The model visualizations were considered good in general. For the Declare view, it was pointed out that the way unary constraints are displayed is *"a bit more intuitive than in all the papers"* (D2) and that the constraint template labels are useful to *"help explain the notation"* (D3). The textual view was considered a useful addition as evident by statements such as *"finally a textual version, I like it"* (D4) and *"it's quite good to have this written in text"* (B3). The automaton view, however, was generally considered to be more *"for theoretical people"* (D4).

Exporting the discovered model takes into account the currently opened model visualization and also the support filters. This means that if the user wants to export a model in *decl* format then the Declare view must be selected and only the parts of the model that are not filtered out with the support sliders are exported. However, not all participants assumed correctly that filters affect the exporting and some of them needed help to export the correct model (obs.).

2) Conformance checking: The implementation of the conformance checking feature in RuM was generally perceived well (*"it is presented in a pleasing way"*, D4, *"I like this conformance checking, well done!"*, D3). Both Declare and BPM experts had similar suggestions, comments and reported on similar issues related to the conformance checking task of our scenario.

One of the main differences of the conformance checking section with respect to the other sections is that it uses two input files (the model and the event log). This was solved in the user interface by treating the model as the main input of the section and the event log file as a parameter. During the evaluation this turned out not to be a problem. Only two participants (B2 and D2) paused for a moment before finding how to select the event log (obs.).

Switching between the original and aligned trace in the Declare Replayer appeared to be difficult for both groups with only B1 not needing any help (obs.). It was not obvious for most participants that "Show Insertions" and "Show Deletions" (Fig. 4) can be used for this purpose (*"I would not have guessed the meaning of that button"*, B2). Most participants needed some time or instructions from the facilitator to make the connection and to set the toggle buttons correctly (obs.). It was also noted that *"showing the deletion is like a negation of a negation"* (D3), which could also have been a source of confusion.

3) Model editor: The main difference between the two studied groups during model editing was that most of the Declare experts (D2, D3, D4) tried at first to edit the model by clicking on the visualization (obs.), which is not possible in RuM. Meanwhile, all the BPM experts used the visualization only to get an overview of the model and did not attempt to modify it (obs.). This could appear counter-intuitive since BPM experts are expected to be used to modifying graphical process models. However, this phenomenon could be related to the fact that the only existing Declare editor is a visual editor [27].

When editing the model, we asked the participants to remove a PRECEDENCE constraint. This turned out to be difficult for some of them (B4, D2, D3), because in the case of PRECEDENCE constraints the activities are presented in reverse order (target activity before the activation activity), when compared to other templates. This issue was specifically pointed out by statements such as *"Is this saying the same thing or is this saying the opposite thing?"* (B4), *"so this is confusing because it's the other way around"* (D2), *"ah right, the activation and target here are swapped"* (D3).

There were also some issues related to the syntax of data conditions. D1 and D2 attempted to use an equal sign instead of the word "is" for equalities (obs.), while B3 commented while entering the condition *"I hope this is the way to write it... is it so English like?"* (B3). D2 suggested adding a help button describing the syntax and D3 suggested that attributes in the model should be recognized by the editor while typing the data conditions.

4) Log generation: For the log generation, there were no noticeable differences between the two groups. Multiple participants (B1, B3, D2, D4) expected "Model Constraints" (Fig. 6) to be somehow functional (since it appears in the parameter panel). For example, B1 explored the constraint list for a bit before asking *"I don't have to put anything here?"* (B1) and B3 clicked multiple times on the constraints while exploring the generated log (obs.). In addition, multiple participants (B2, B3, B4, D4) attempted to click on the generated events when checking if an event was generated with the attributes required by the model (obs.). Instead, in the application, the attributes of the events in a trace can be shown all together using a toggle button (Fig. 6).

VI. THREATS TO VALIDITY AND CONCLUSION

In this paper, we presented RuM, a comprehensive toolset for rule mining. Our evaluation provides indication that the tool is usable for novice and expert users. Novice users were particularly satisfied about the new look of Declare and about the effort made to improve the understandability of Declare

models, while Declare experts were more appreciating the fact that RuM collects the most widespread Declare-based process mining techniques in a single tool.

The goal of the study was to collect feedback and to evaluate the usability of RuM for individuals with different backgrounds. It was thus reasonable to conduct an in-depth qualitative study with selected participants from diverse backgrounds related to their knowledge and experience with Declare. Conducting a study with a small sample of participants is common because research has shown that the number of additional insights gained deteriorates drastically per participant [28].

There are, however, some threats to validity associated with this study design. First, we evaluate the use of a specific tool by specific individuals in a specific context over a limited period of time. Despite carefully selecting the participants and creating a setting that is close to how the tool would be commonly used, it is not possible to generalize our findings beyond our study context. In addition, the study was conducted by a team of researchers that might potentially interpret findings differently. We attempted to mitigate this threat by ensuring that observations, interviews and the analysis of the obtained data was collaboratively conducted. We also abstained from making causal claims, instead providing a rich description of the behavior and reported perceptions of participants.

For future work, we plan to continue developing RuM based on the feedback received during the user evaluation, focusing in particular on aspects that proved to be the most critical ones. Additionally, we plan to explore the feasibility of developing a visual editor for MP-Declare models, so that elements can be added/deleted/modified in the graphical view directly. We also plan to add an inventory of files that can be easily retrieved and used throughout all the sections of the tool. This will facilitate the construction of pipelines based on the analysis instruments provided by the application. Another avenue for future work is adding support for online analysis of event logs based on Declare taking inspiration from the works presented in [29] concerning online discovery and in [30] for process monitoring.

Acknowledgements. The authors would like to thank all the participants of the user evaluation for taking the time to evaluate RuM and for providing us with invaluable feedback on how RuM can be improved in the future. The work of A. Alman was partly supported by the Estonian Research Council (project PRG887). The work of C. Di Ciccio was partly supported by MIUR under grant "Dipartimenti di eccellenza 2018-2022" of the Department of Computer Science at Sapienza University of Rome.

References

[1] L. Fonseca and J. P. Domingues, "ISO 9001:2015 edition- management, quality and value," *International Journal for Quality Research*, vol. 1, no. 11, pp. 149–158, 2017.

[2] W. M. P. van der Aalst, "Process mining: Overview and opportunities," *ACM Trans. Manage. Inf. Syst.*, vol. 3, no. 2, Jul. 2012.

[3] ——, *Process Mining - Data Science in Action, Second Edition*. Springer, 2016.

[4] 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," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 4, pp. 686–705, 2019.

[5] T. Slaats, "Declarative and hybrid process discovery: Recent advances and open challenges," *Journal on Data Semantics*, vol. 9, no. 1, pp. 3–20, Mar 2020.

[6] C. Haisjackl, I. Barba, S. Zugal, P. Soffer, I. Hadar, M. Reichert, J. Pinggera, and B. Weber, "Understanding declare models: strategies, pitfalls, empirical results," *Software and Systems Modeling*, vol. 15, no. 2, pp. 325–352, 2016.

[7] A. Burattin, F. M. Maggi, and A. Sperduti, "Conformance checking based on multi-perspective declarative process models," *Expert systems with applications*, vol. 65, pp. 194–211, 2016.

[8] W. M. P. van der Aalst and M. Pesic, "DecSerFlow: Towards a truly declarative service flow language," in *WS-FM*, 2006, pp. 1–23.

[9] G. De Giacomo, R. De Masellis, and M. Montali, "Reasoning on LTL on finite traces: Insensitivity to infiniteness," in *AAAI*, 2014, pp. 1027–1033.

[10] C. Di Ciccio, M. L. Bernardi, M. Cimitile, and F. M. Maggi, "Generating event logs through the simulation of declare models," in *EOMAS*, 2015, pp. 20–36.

[11] F. Mannhardt and D. Blinde, "Analyzing the trajectories of patients with sepsis using process mining," in *RADAR+ EMISA@ CAiSE*, 2017, pp. 72–80.

[12] F. Mannhardt, M. de Leoni, H. A. Reijers, W. M. P. van der Aalst, and P. J. Toussaint, "Guided process discovery - A pattern-based approach," *Inf. Syst.*, vol. 76, pp. 1–18, 2018.

[13] A. Dix, J. Finlay, G. D. Abowd, and R. Beale, *Human-computer interaction*. Pearson Education, 2003.

[14] B. Myers, S. E. Hudson, and R. Pausch, "Past, present, and future of user interface software tools," *ACM Trans. Comput.-Hum. Interact.*, vol. 7, no. 1, pp. 3—-28, Mar. 2000.

[15] J. Nielsen, "Ten usability heuristics," 1994, https://www.nngroup.com/.

[16] C. W. Gunther and H. M. W. Verbeek, *XES - standard definition*, ser. BPM reports. BPMcenter.org, 2014, vol. 1409.

[17] V. Skydanienko, C. Di Francescomarino, C. Ghidini, and F. M. Maggi, "A tool for generating event logs from multi-perspective declare models," in *BPM (Dissertation/Demos/Industry)*, 2018, pp. 111–115.

[18] F. M. Maggi, C. Di Ciccio, C. Di Francescomarino, and T. Kala, "Parallel algorithms for the automated discovery of declarative process models," *Inf. Syst.*, vol. 74, pp. 136–152, 2018.

[19] C. Di Ciccio and M. Mecella, "On the discovery of declarative control flows for artful processes," *ACM Trans. Manag. Inf. Syst.*, vol. 5, no. 4, Jan. 2015.

[20] V. Leno, M. Dumas, F. M. Maggi, M. L. Rosa, and A. Polyvyanyy, "Automated discovery of declarative process models with correlated data conditions," *Inf. Syst.*, vol. 89, p. 101482, 2020.

[21] 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*, 2012, pp. 82–97.

[22] A. Alman, K. J. Balder, F. M. Maggi, and H. van der Aa, "Declo: A chatbot for user-friendly specification of declarative process models," in *BPM Demos*, 2020.

[23] J. Brooke, "SUS: a 'quick and dirty' usability scale," *Usability evaluation in industry*, p. 189, 1996.

[24] A. Bhattacherjee, "Understanding information systems continuance: an expectation-confirmation model," *MIS quarterly*, pp. 351–370, 2001.

[25] K. Holtzblatt, J. B. Wendell, and S. Wood, *Rapid contextual design: a how-to guide to key techniques for user-centered design*. Elsevier, 2004.

[26] P. T. Kortum and A. Bangor, "Usability ratings for everyday products measured with the system usability scale," *International Journal of Human–Computer Interaction*, vol. 29, no. 2, pp. 67–76, 2013.

[27] M. Westergaard and F. M. Maggi, "Declare: A tool suite for declarative workflow modeling and enactment," in *BPM Demos*, 2011.

[28] J. Nielsen and T. K. Landauer, "A mathematical model of the finding of usability problems," in *INTERCHI*, 1993, pp. 206–213.

[29] A. Burattin, M. Cimitile, F. M. Maggi, and A. Sperduti, "Online discovery of declarative process models from event streams," *IEEE Trans. Serv. Comput.*, vol. 8, no. 6, pp. 833–846, 2015.

[30] F. M. Maggi, M. Montali, M. Westergaard, and W. M. P. van der Aalst, "Monitoring business constraints with linear temporal logic: An approach based on colored automata," in *BPM*, 2011, pp. 132–147.

Process Mining Meets Causal Machine Learning: Discovering Causal Rules from Event Logs

Zahra Dasht Bozorgi*, Irene Teinemaa†, Marlon Dumas‡, Marcello La Rosa*, Artem Polyvyanyy*
*University of Melbourne
zdashtbozorg@student.unimelb.edu.au
{marcello.larosa, artem.polyvyanyy}@unimelb.edu.au
†Booking.com
irene.teinemaa@booking.com
‡University of Tartu
marlon.dumas@ut.ee

***Abstract*—This paper proposes an approach to analyze an event log of a business process in order to generate case-level recommendations of treatments that maximize the probability of a given outcome. Users classify the attributes in the event log into controllable and non-controllable, where the former correspond to attributes that can be altered during an execution of the process (the possible treatments). We use an action rule mining technique to identify treatments that co-occur with the outcome under some conditions. Since action rules are generated based on correlation rather than causation, we then use a causal machine learning technique, specifically uplift trees, to discover subgroups of cases for which a treatment has a high causal effect on the outcome after adjusting for confounding variables. We test the relevance of this approach using an event log of a loan application process and compare our findings with recommendations manually produced by process mining experts.**

***Index Terms*—process mining, causal ML, uplift modeling**

I. Introduction

A business process is a collection of events, activities, and decisions that collectively lead to an outcome that is of value to a customer [1]. Some outcomes are value-adding (e.g. the customer is satisfied with the delivery of a product) while others are not (e.g. a customer submits a complaint). Naturally, organizations strive to maximize the positive outcome rate of their processes or, conversely, to minimize the error rate.

Process mining techniques allow one to analyze the executions of a process to uncover sources of negative outcomes. Existing process mining techniques, such as [2]–[5], are geared towards identifying *correlation* between observational data and outcomes (e.g. cases where the customer is satisfied have less rework loops) rather than *causation* (e.g. customers who submit incorrect details cause more rework loops, leading to lower satisfaction). Meanwhile, causal inference techniques allow one to discover and measure causal relations between *treatments* (e.g. checking the customer data) and *outcomes* (e.g. the customer is satisfied) both from randomized experiments and from observational data.

Recently, a family of techniques, namely causal machine learning, have emerged, which make use of machine learning methods to analyze causal effects. Causal machine learning encompasses techniques for estimating the causal effect of a treatment on an outcome given a set of potentially confounding variables (average treatment effect estimation) as well as techniques for classifying samples in a population based on the incremental effect of applying a treatment versus not applying it with respect to an outcome (uplift modeling).

In this study, we leverage these techniques to address the following question: Given a set of treatments (each with a certain cost), which may affect a business process outcome (with a certain benefit), which treatments yield the highest causal effect on the outcome and to which subset of cases should they be applied? In line with this, the contribution of this paper is an approach to:

- discover case-level treatment recommendations to increase the positive outcome rate of a process;
- identify subsets of cases to which a recommendation should be applied;
- estimate the causal effect and the incremental Return-on-Investment (ROI) of a treatment.

The approach is designed to require minimal input from users. Users specify which attributes in the event log are controllable, meaning that their value can be manipulated by process participants, i.e. the employees who perform the various process tasks. Setting the value of a controllable attribute corresponds to a treatment. For example, in an order-to-cash process, setting an attribute *discountGranted* to true means that a discount was granted. The attributes capturing such treatments may be derived during log pre-processing from the presence or absence of certain tasks, e.g. *discountGranted* may be derived from the presence of task "Grant Discount".

Given this input, we apply a technique to discover precondition-treatment-outcome rules with high support. Since neither a rule's support nor its confidence imply causation, we use a causal machine learning technique to assess the causal effect of the rule and to discover subsets of cases for which the treatment has the highest incremental success probability (*uplift*). We then select the rules with the highest uplift. We report on a validation of this approach using a log of a process mining challenge (BPIC 2017) and compare the findings of our

978-1-7281-9832-3/20/$31.00 ©2020 IEEE
DOI 10.1109/ICPM49681.2020.00028

approach against those reported in the entries of this challenge.

The structure of the paper is as follows. We discuss related work in Section II. We introduce preliminary concepts in Section III, describe our approach in Section IV and present the validation in Section V. In Section VI we conclude the paper and discuss future work.

II. Related Work

Previous work has shown that one can rely on influence analysis to identify improvement opportunities from event logs [2], [3]. In [4], rules to describe root causes of anomalous process cases are extracted, while [5] relies on classical data mining methods to discover key attributes for root-cause analysis. These methods identify correlations between attributes and the outcome but do not test for causality.

The problem of discovering cause-effect relations is addressed in [6]. This approach performs time series analysis to identify causal relations. This is different from our approach as the analysis is done at the process-level, while we provide case-level recommendations. The work in [7] studies process-level factors that impact outcomes but fails to determine causalities between the two. A manual approach for confirming pre-identified causal relationships was proposed in [8]. It uses structural causal models to confirm cause-effect assumptions, control the effects of confounding, and answer counterfactual questions about the process.

Polyvyanyy *et al*. [9] present a (semi-)automated approach, called *causality mining*, to discover causal dependencies between events in large arrays of data. The discovery is based on a notion of proximity of events in terms of time, space, and semantics. The level of automation depends on the availability of the formalized domain knowledge.

In summary, previous work either addresses the problem of finding correlation rather than causation between factors and outcomes, or causation is addressed at the process-level, not at the case level. Moreover, previously identified causal effects either need to be confirmed manually, or extensive domain knowledge is required as input.

III. Preliminaries

In this section we formalize preliminary concepts that are required to describe our approach, such as event logs, action rule mining, causal inference and uplift trees.

A. Event Logs

Process mining studies methods for improving real-world processes based on event data [10]. These data are often available in the form of an *event log*. Event logs contain records of completed cases of a process. Each case is a record of the execution of a particular process instance and consists of a number of events. Each event has three mandatory attributes: (1) the case identifier indicating which case that event belongs to, (2) the activity name specifying the related activity for that event, and (3) the timestamp, showing when the event occurred. In addition, an event may have attributes, such as the resource carrying out the related activity. An event is formally defined as follows:

Definition 1. *(Event)* An *event* is a tuple $(a, c, t, \langle (d_1, v_1), \ldots, (d_m, v_m) \rangle)$, where a is an activity name, c is a case ID, t is a timestamp, and $(d_1, v_1), \ldots, (d_m, v_m)$, $m \in \mathbb{N}$, are event attribute name-value pairs. Given an event e, c_e denotes the identifier of the case.

A trace is a sequence of events that captures the execution of one case of a business process.

Definition 2. *(Trace)* A *trace* σ is a finite sequence of events $\langle e_1, \ldots, e_n \rangle$, such that $\forall\, i, j \in [1..n], c_{e_i} = c_{e_j}$, i.e., all events in the trace refer to the same case.

Given the above, an event log is defined as a set of traces.

Definition 3. *(Event Log)* Let $\mathcal{E}$ be the universe of events. An *event log* is a set $L \subset \mathcal{E}^*$.

B. Action Rule Mining

Action rule mining is an extension of classification rule discovery [11]. While a classification rule predicts the class label of a data object, an action rule suggests what attribute values should be changed to increase the likelihood of that object being re-classified to another group. In [11], action terms and action rules are defined as:

Definition 4. *(Atomic Action Terms)* An *atomic action term* is an expression $(a : a_1 \rightarrow\ a_2)$, where a is an attribute and a_1 and a_2 are possible values attribute a.

If $a_1 = a_2$ then a is *uncontrollable*, denoted by $(a : a_1)$.

Definition 5. *(Action Terms)* A set of *action terms* is the smallest set such that: 1. If t is an atomic action term, then t is an action term. 2. If t_1 and t_2 are action terms, then $t_1 \wedge\ t_2$ is an action term. 3. If an action term t contains atomic action terms $(a : a_1 \rightarrow\ a_2)$ and $(b : b_1 \rightarrow\ b_2)$, then $a \neq b$.

Definition 6. *(Action Rules)* An *action rule* is an expression $r = [t_1 \Rightarrow\ t_2]$, where t_1 is an action term and t_2 is an atomic action term.

C. Causal Inference

Causal inference is concerned with determining the causal effect between a treatment and an outcome [12]. Suppose that we have a treatment A and an outcome Y. A potential outcome Y^a is the outcome that *would* be observed if the treatment was set to $A = a$. Focusing on a single treatment, each individual in the population of interest has two potential outcomes, $Y^{a=1}$ for being treated and $Y^{a=0}$ for being untreated. With these definitions, the average treatment effect (ATE) is defined as:

Definition 7. *(Average Treatment Effect)* Suppose we have a population of individuals. Let $\mathbb{E}[Y^{a=1}]$ be the average outcome if the whole population receives the treatment and $\mathbb{E}[Y^{a=0}]$ be the average outcome if the entire population does not receive the treatment.

$$ATE : \tau = \mathbb{E}[Y^{a=1} - Y^{a=0}]$$

Researchers are often interested in estimating the Conditional Average Treatment Effect (CATE), which is the expected treatment effect for a subgroup in the population. It enables personalizing treatments and leads to a better understanding of causal mechanisms [13].

Definition 8. *(Conditional Average Treatment Effect)* Suppose that X is a set of covariates characterizing the subgroup of interest.

$$CATE : \tau(x) = \mathbb{E}[Y^{a=1} - Y^{a=0}|X = x]$$

Measuring the causal effect would be straightforward if we knew both potential outcomes for each individual. However, in the real world, we can only observe one outcome for each individual in the population, corresponding to the treatment that the user actually received. To identify causal effects from observational data, three conditions must be met: *Exchangeability*, *Positivity*, and *Consistency*. Exchangeability (also known as ignorablility) means that given pre-treatment covariates X, treatment assignment is independent of the potential outcomes.

$$Y^1, Y^0 \perp\!\!\!\perp A|X$$

The consistency assumption states that the potential outcome under treatment $A = a$ is equal to the observed outcome if the actual treatment received is $A = a$.

$$Y = Y^a \text{ if } A = a \text{ for all } a$$

Finally, the positivity assumption states that for every set of values for X, treatment assignment is not deterministic. This means that every population of interest has some chance of getting either treatment.

$$P(A = a|X = x) > 0 \text{ for all } a \text{ and for all } x$$

In practice, the most problematic of these three conditions is exchangeability. One approach to ensure that this condition is met is to conduct a randomized experiment (also known as an A/B test), where the treatment is assigned randomly to each individual. Randomization of treatment assignment ensures that the treated and untreated groups are exchangeable, so that the causal effect can be consistently estimated from the observed data. However, conducting a randomized experiment is not always possible, since it might be expensive, time-consuming, or unethical. In these cases, the best we can do is to carry out an observational study. In observational studies, treatment is often not randomized, meaning that the characteristics of the treated group might be different from the untreated. If the treatment assignment is not independent of the potential outcomes, then there exists a set of variables that affect both treatment and outcome. This is known as a confounder. Fig. 1 depicts the causal relations between a treatment A, an outcome Y, and a shared cause (i.e. a confounder) L.

To estimate causal effects from observational data, we need to control confounding. To this end, we need to identify a set of variables, such that adjusting for these variables would make the exchangeability assumption hold. If some of the adjustment variables cannot be observed in the data, then causal effects are not identifiable in the observational study.

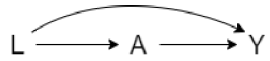


Figure 1: Causal DAG depicting a confounding effect.

D. Uplift Tree

Uplift modeling is concerned with estimating the causal effect of an action on the outcome of a particular instance (e.g. customer) [14]. In other words, the aim is to estimate the change in class probabilities caused by an action. This is different from conventional prediction problems where a model is used to predict an outcome. For example, consider a marketing campaign. A conventional classifier would predict which customers will buy a product after a marketing action is performed without taking into account whether these users would have bought the product if the marketing action had not taken place. However, marketers are actually interested in finding the customers who are most likely to buy something *because of* the marketing action. This is what uplift modeling is trying to achieve, which amounts to identifying subsets of instances with a high $CATE$.

In this study, we apply uplift modeling to business processes. We seek to estimate the change in the outcome of a process instance because of an action being applied to that case. Many uplift modeling approaches exist in the literature. We use the method proposed in [14] to discover subgroups in the event log where a proposed treatment works best. It is a tree-based algorithm where the splitting criterion is designed to maximize the difference in $CATE$. The splitting criterion is the following:

$$D_{gain} = D_{AfterSplit}\left(P^T(Y) : P^C(Y)\right) - D_{BeforeSplit}\left(P^T(Y) : P^C(Y)\right),$$

where D can be substituted by the KL-divergence, squared Euclidean distance, or the chi-squared divergence and $P^T(Y)$ and $P^C(Y)$ are the probability distributions of the outcome in the treatment and control groups, respectively.

IV. Approach

Our approach consists of three steps as shown in Fig. 2. First, candidate treatments are generated using action rule mining. Next, we identify subgroups in the population for every candidate treatment using uplift trees. Finally, we present a cost-benefit model to rank the rules based on the benefit of a positive outcome, the cost of treatment and its uplift.

A. Candidate Treatments Identification

This step requires the user to provide an event log, the controllable and uncontrollable attributes, the outcome variable, and a minimum support. The classification of attributes into controllable and uncontrollable ensures that the candidate treatments are actionable, meaning that no change in the uncontrollable attributes is suggested by the action rules. In

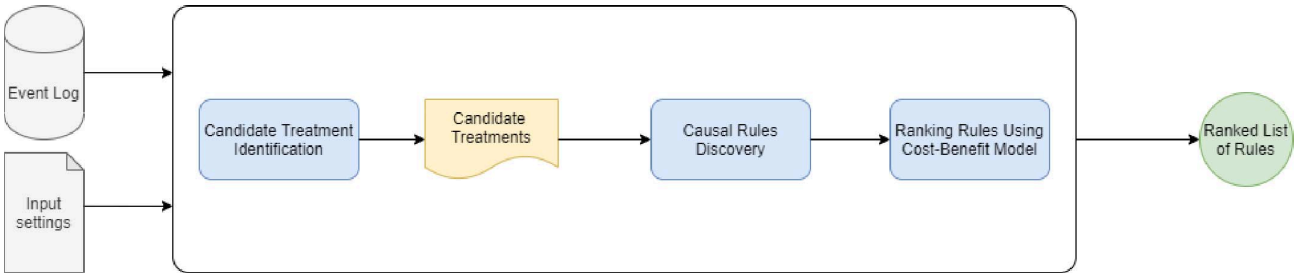


Figure 2: Overview of the approach.

action rule mining, candidate treatment extraction is based on support. We seek to obtain rules that are likely to generate high revenue, which implies that the treatment should be related to the effect for a sufficiently large sub-population of cases. This is achieved by the support threshold. For example, the user may decide that a candidate treatment should be linked to the effect in at least 2% of cases (support threshold); otherwise, the treatment is discarded.

Fig. 3 shows an example action rule. It was obtained based on the BPI Challenge 2017 event log using the method described in [11]. The rule states that in cases where the customer's credit score is low, changing the number of terms from the interval 6–48 months to 97–120 months will increase the likelihood of the outcome variable (Selected) to change from negative (0) to positive (1). The rule is supplied with the support and confidence measurements. In the rule, *CreditScore* is an example of an uncontrollable attribute, while the number of terms is considered controllable because the company can take steps to reduce or increase it. In our method, however, we only use the treatment part of the action rule. This is because action rules are an extension of classification rules, and they identify associations rather than causation. We seek to find a sub-population where the treatment causes the desired outcome. Therefore, in the next step, we use uplift modeling to discover *causal* rules.

B. Causal Rules Discovery

This step aims to discover subgroups X for which a certain treatment A has a high positive causal effect on the outcome Y. To this end, for each candidate treatment, we perform steps detailed below.

First, we build an uplift tree and take rules with high *uplift*:

$$Pr(Y = 1|A = 1, X = x) - Pr(Y = 1|A = 0, X = x).$$

A popular but unjustified belief regarding uplift from observational data is that it can be estimated using the above formula. Uplift cannot be estimated this way unless we assume that for each individual in the population, A is independent of the counterfactual outcomes Y^1 and Y^0 conditional on X (exchangeability assumption) [15]. This assumption holds only when treatment assignment is randomized. In randomized controlled trials, treatment is randomized by design. However, in observational studies, the treated and the untreated individuals are systematically different. The advantage of using the uplift tree method is that we can address the above issue by using its normalization feature. Normalization punishes tests that split the treatment and control groups in different proportions. These splits indicate situations where the test is not independent of the group assignment, and thus, the exchangeability assumption is violated. For a test A and the KL-divergence criterion, normalizing factor is calculated as follows:

$$I(A) = H\left(\frac{N^T}{N}, \frac{N^C}{N}\right) KL\left(P^T(A) : P^C(A)\right) + \frac{N^T}{N} H\left(P^T(A)\right) + \frac{N^C}{N} H\left(P^C(A)\right) + \frac{1}{2},$$

where N^T and N^C are the numbers of cases in the treatment group and the control group, respectively, and $N = N^T + N^C$ is the total number of cases. For the squared Euclidean distance and the Chi-squared divergence, entropy is replaced by the Gini index. The first term of this factor punishes tests with imbalanced splits and, therefore, adjusts for confounding effects. The following two terms prevent bias towards tests with high numbers of outcomes. After normalization, the final splitting criterion is the gain divided by the normalizing value.

While the normalization factor adjusts for biases related to observed confounders provided to the uplift tree as input variables, it does not ensure that there are no unobserved confounders that could invalidate the interpretation of the uplift estimates as true causal effects. As an optional step, to strengthen the validity of the study, a manual check for confounding effects can be carried out by specifying a causal graph (depicting both observed and unobserved confounders) and identifying whether a valid adjustment set exists using the back-door criterion as described in [16].

Fig. 4 shows an example uplift tree. The sub-populations of interest are in the leaves of the tree, and the user may pick the leaves that have an uplift score above a certain threshold.

C. Ranking Rules Using a Cost-Benefit Model

An uplift score of an identified rule quantifies the causal effect of applying each intervention. However, it is not always profitable to carry out interventions with a negative incremental return-on-investment (ROI). So, we incorporate the uplift measure in a cost-benefit model that corresponds to the incremental ROI. Knowing the expected causal effect (estimated in the previous step) of applying a treatment to the discovered sub-population, the value of the desired outcome,

Rule : $r = [(CreditScore : low) \wedge (NoOfTerms : [6-48] \rightarrow [97-120])] \Longrightarrow [Selected : 0 \rightarrow 1]$,
with support 0.057 *and confidence* 0.764;

Figure 3: Example of an Action Rule.

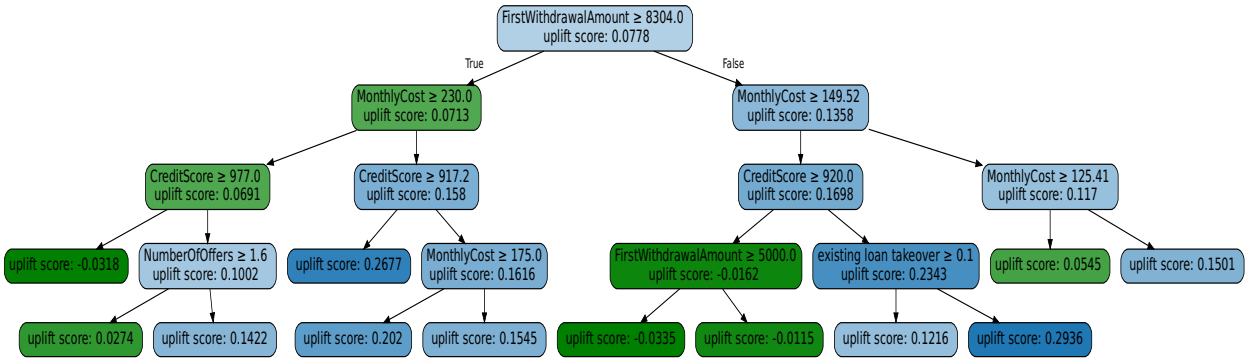


Figure 4: Example of an uplift tree.

and the costs of the treatments, we produce a cost-aware ranking of the rules. According to [17], there are two types of treatment costs: 1) Fixed impression cost, i.e., the cost that occurs when applying the treatment, such as the cost of a phone call, 2) Triggered cost, i.e., the cost that occurs only if the treated case reaches a positive outcome such as lowering the interest rate of a loan. In the following, we assume that the triggered costs are not present.

We use this notation to define the cost-benefit model:

- v: value (benefit) of a positive outcome, assuming that it is constant and given as prior knowledge;
- c: impression cost for a treatment;
- u: uplift of applying a treatment; and
- n: size of the treated group.

We define the net value of applying a treatment is as follows:

$$net = n \times (u \times v - c).$$

Note that $n \times u \times v$ represents the incremental value of the treatment and $n \times c$ the incremental cost.

V. Evaluation

The proposed approach was implemented in Python 3.7 using the ActionRules[1] package for generating actionable recommendations and the CausalML package [18] for constructing uplift trees. The relevance of the approach is shown through a case study using the BPI Challenge (BPIC) 2017 log.[2] We chose this log among all other BPI Challenge logs because the approach requires a setting where an outcome can be influenced by interventions that can take the form of a change in the case attributes. The BPIC 2017 log was the only one that satisfies these criteria. Since we did not have access to a subject matter expert from the company that provided the log, we compare our results with the reports of the winners of the challenge. The main goal of this experiment is thus to compare the recommendations that we generated automatically with those that the winners of the challenge produced.

A. Dataset

The BPI 2017 log records cases of a loan application process at a Dutch financial institute, which were filed in 2016 and handled up until 2 February 2017. It contains 31,509 applications (cases), 1,202,267 events and 42,995 offers. In addition to the attributes found in the log, we engineered the number of offers made to the customer as an additional feature. We also filtered the log to remove cases where the value of the outcome variable was missing.

For each application one or more offers may be created but the customer may only select one offer. In many cases, the customer does not select any of the bank offers. So the target variable in this study is the attribute 'Selected'. It is a Boolean attribute which is equal to true if the customer selects an offer and false otherwise.

Next, we classified the other attributes into controllable and uncontrollable. These features were considered uncontrollable:

- Application type (new credit or limit raising)
- Loan goal (reason for the loan application)
- Customer credit score
- Requested amount.

The following features were classified as controllable:

- Number of offers
- Number of payback terms (months)
- Monthly cost
- Initial withdrawal amount.

[1]https://github.com/lukassykora/actionrules
[2]doi:10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b

B. Results

We ran the action rule discovery algorithm on the above dataset with support = 3 and confidence = 55, resulting in 24 rules containing 17 distinct recommendations. For each recommendation we constructed an uplift tree to find the sub-population for each actionable recommendation using the following settings:

- maximum depth of the tree = 5
- minimum number of samples for a split = 200
- minimum number of samples in the treatment group for a split = 50
- regularization parameter = 100
- evaluation function = Kullback-Leibler divergence.

We extracted the following rules:

- Action 1: Decreasing the initial withdrawal amount from 7,500–9,895 to 0–7,499. Sub-population: Limit raising customers with a credit score greater than 885 and a monthly cost below 120 Euros.
- Action 2: Increasing number of terms from 6–48 months to more than 120 months. Sub-population: Customers whose credit scores are between 899 and 943 and their first withdrawal amount is less than 8,304.
- Action 3: Increasing number of terms from 6–48 months to 61–96 months. Sub-population: Customers whose loan goal is not existing loan takeover, have a credit score less than 920, their offer includes a monthly cost greater than 149 and the first withdrawal amount is less than 8304.
- Action 4: Increasing number of terms from 6–48 months to 97–120 months. Sub-population1: Customers with a credit score less than 982, first withdrawal amount greater than 8,304, and a monthly cost between 154 and 205. Sub-population2: Customers with a credit score between 781 and 982, first withdrawal amount less than 8,304, and a monthly cost greater than 147.
- Action 5: Decreasing first withdrawal amount from 7,500–9,895 to 5,750–7,499 and decreasing number of terms if greater than 120 months to 49–60 months. Sub-population: Customers with an offer that has a monthly cost less than 150.
- Action 6: Decreasing first withdrawal amount from 7500–9895 to 0–7,499 and decreasing number of terms if greater than 120 months to 97–120 months. Sub-population: New credit applicants with a credit score less than 914 that have an offer with a monthly cost more than 150.
- Action 7: Increasing first withdrawal amount from 7,500–9,895 Euros to 9,896–75,000. Sub-population: New credit application where the loan goal is existing loan takeover and the customer credit score is 825.
- Action 8: Decreasing first withdrawal amount from 9,896–75,000 to 1,490–7,499 and increasing number of terms from 49–60 to 97–120. Sub-population: Customers with credit score lower than 933 and monthly cost greater than 154.

C. Discussion

The BPI challenge had three categories: student, professional and academic. Below, we discuss the recommendations of the winners in each category and compare them with our findings. Since our recommendations are at case level, we only discuss the case-level recommendations in these reports.

1) Academic: The winning report in this category [19] starts by producing the as-is process model of the underlying loan application process, and then identifies the variants of the process in order to better understand the data at hand. The authors analyze the process outcome and make the recommendation to decrease the monthly cost or increase the number of terms to more than 120. This is very similar to our Action 2 in the previous section. However, we found that this rule should be applied to customers in a specific range of credit score (between 899 and 943) and first withdrawal amount (less than 8304). It is for this specific sub-population that this action has a high incremental effect. For example, in the rule's uplift tree, it is indicated that if the first withdrawal amount is higher than 10,000 the uplift is only 8%. So while it might generally be beneficial to increase the number of terms, applying this action might not lead to an increase in revenue all the times.

They also carried out an analysis on the application type. There are two application types in this dataset: new credit and limit raise. The majority of the applications are new credit applications (89%). They found that applications for limit raising have a higher rate of success than new credit applications. Limit raising applications are included in a sub-population in only one rule. One possible reason for this is that this type of applications already have a high likelihood of being successful. So, applying a treatment would be unnecessary. In addition, we have found that more than the application type, it is the credit score that determines the causal effect of an action on the case outcome. According to our results, credit score is included in almost all the rules, but the application type is present in less than half the rules.

Regarding the number of offers, the authors of the report observed that there is an association between having more offers and the customer not cancelling the application. The candidate treatment identification part of our method was not able to recommend any action regarding the number of offers made to the customer. This is because the action rules method generates the rules based on support. In this dataset, one offer was created for the majority of applications. Therefore, any rule with higher number of offers would not reach the minimum support threshold.

2) Professional: Similar to the academic category, in this report [20] the authors discovered the association between the number of offers and a successful outcome and made the recommendation to increase the number of offers. They also analyzed different loan goals and reached the conclusion that it would be beneficial to improve the instructions on the required documentation. This is to decrease the duration of the application. They were able to identify the cause of the delays in the application process by manually checking the

time needed to validate the application and the number of times requests for document completions are sent.

Further, they analyzed the impact of credit score on the customer's decision and found that high credit customers have significantly higher chance of a successful outcome. This is in line with our finding: Actions 2–4, 6–8 are concerned with customers with low credit score, meaning that there is no need for change in applications with high credit score.

The authors also performed predictive analysis to determine which attributes have an impact on the outcome. They found that "credit score has a significant impact on the decision of the customer to take the offer or not". They state that this might be because the bank has more competitive offers for higher credit customers and they recommend the bank to further investigate this issue. This is in line with our finding that most of the discovered sub-groups in our rules include lower credit score customers who are more in need of treatments. However, we go beyond the finding in the report and provide concrete actionable rules for customers that have lower credit scores and are more likely to reject their offers.

Finally, the authors of the winning report in the Professional category found that all other variables being equal, raise limit applications are more likely to be successful than new credit applications. However, this finding is based on application type having a low p_value which indicates association rather than causation. Furthermore, they found that monthly cost and duration also impact the outcome. Regarding the monthly cost, this is related to the number of terms. So if a rule recommends that the number of terms should be increased, it is implicitly recommending the monthly cost to be decreased. The duration of an application depends both on what the client nominated in their loan application and what the bank has approved. It also depends on the time it takes for the client to respond to the bank's requests (e.g. requests for additional documents) and the time for the bank to process the application. For the above reasons, these are spurious results and as such did not emerge in our findings.

3) Student: In this report [21], the authors provide five recommendations:

- Send offers to clients as soon as possible.
- Maximize the number of terms.
- Minimize the first withdrawal amount.
- Minimize the monthly cost.
- Minimize the time intervals between sending the offers to the client.

The authors found that if offers are sent within four days of submitting the application, cancellation rate may decrease between 5% and 10%. Moreover, they found that in the cases where multiple offers where made, keeping the mean time between the offers to below four days may decrease the cancellation rate by 2% to 5%. Our recommendations do not include the timing of the offers because it requires additional feature engineering. Since we did very minimal feature engineering to keep the method as automated as possible, we cannot comment on the authors' first and last recommendations.

Regarding the number of terms, they found that by keeping it to above 60, they can decrease the cancellation rate by 3% to 9%. We also identified three rules regarding the number of terms (Actions 2–4 and 8). This recommendation is generally made for all the customers of the bank. However, we found that the magnitude of the increase has different causal effects depending on the circumstance of each case. For instance, suppose we have a customer with a credit score of 900. According to our Action 2 if the first withdrawal amount for this customer was determined to be less than 8,304, then the number of terms for this customer should be increased to more than 120. However, if the same customer has an offer with the first withdrawal amount greater than 8,304 and a low monthly cost, then the increase in the number of terms to the interval 97–120 is sufficient. Additionally, an overall increase in the number of terms will not necessarily be beneficial. For example, according to the uplift tree for our Action 3 (see Fig. 4), increasing the number of terms for clients with their loan goal being that of existing loan takeover, has a low uplift score. This means that for these types of applicants, offers with a high number of terms will not influence the outcome. It should also be noted that we have discovered two rules (Actions 5 and 6) where decreasing the number of terms is recommended. But only when it is paired with decreasing the first withdrawal amount. Rule 5 describes situations where the offer has a high initial withdrawal amount and a low monthly cost. So it recommends a more balanced offer where the customer can pay less initially but is charged slightly more monthly. Rule 6 describes a similar situation, but for bigger loans (e.g., when monthly cost is higher than 150).

According to the same report, the first withdrawal amount should be less than 5,000 Euros. By keeping it below this amount, a decrease of cancellation by 9% to 12% can be reached. We also found that in most of our rules, the initial withdrawal amount should be decreased. Only in one rule (Action 7) we found that it should be increased, namely for cases where application type is new credit, loan goal is existing loan takeover and the customer's credit score is 825. This might be due to the fact that in such applications, the customer does not have a high enough credit to demand a better offer and at the same time they may not want to increase the number of payback months, because they have already been paying for a previous loan. So, a high initial withdrawal amount might be attractive to such clients.

The authors' recommendation regarding the monthly cost is to minimize it. They found that having a monthly cost below 400 Euros will bring the cancellation rate down by 3% . The monthly cost is closely related to the loan amount and the number of terms. With the loan amount being fixed, increasing the number of terms will automatically result in the monthly cost being decreased. So in those rules where we recommend to increase the number of terms (Actions 2–4 and 8), we are implicitly recommending the monthly cost to be decreased.

D. Threats to validity.

The above validation comes with the usual threat to external validity associated with case studies (lack of generalizability). We do not claim that the approach can discover relevant rules in other contexts. The validation also comes with threats to internal validity. The results may be affected by data quality issues in the event logs as well as misinterpretations of the semantics of the data or of the usefulness of the discovered rules. The latter threat is mitigated by the fact that we cross-checked our interpretation of the data and the results against the reports of the winners of the BPI Challenge who, at the time the contest took place, were able to indirectly resolve doubts about the data and its semantics with domain experts. To compensate for these threats, we provide a software artifact to enable other researchers to reproduce our experiments and to replicate them in other case studies (see link at the end of the paper). The fact that the experiments are based on observational data creates a potential threat to construct validity. The estimated uplift scores may not match those observed when the rules are deployed in practice. A rigorous A/B test should be conducted prior to deploying the recommendation rules in an operational setting.

VI. Conclusion and future work

We proposed an approach to analyze event logs in order to generate recommendations for applying treatments during the execution of a case so as to maximize the probability of an outcome. The approach leverages causal machine learning techniques to estimate the causal effect of a treatment and to identify subsets of cases for which a treatment has the highest incremental effect (uplift). We sketched how the incremental ROI of applying a treatment may be derived from the uplift.

We validated the approach by applying it to the event log of the BPI Challenge 2017. Since we did not have access to a domain expert from the company that supplied the log, we compared our findings to those of the winners of the challenge. We found that most of the generated recommendations matched those of the winners. Furthermore, for each recommendation, our approach identified specific subsets of cases (e.g. specific types of loan applications or customers) for which the treatment could be most effective.

The approach adjusts the estimation of the causal effect of a treatment against the variables extracted from the event log (e.g. case and event attributes). However, it does not adjust for exogenous confounding effects not explicitly captured in the log. An avenue for future work is to extend the approach with a method to identify contextual variables, such as time of the day, geographic location of the process stakeholders, weather, and to validate the discovered causal relations with respect to such variables, for example using structural models as in [8].

The reported validation is preliminary and does not involve feedback from potential users of the technique (managers, analysts) or external validation for example via A/B testing of the identified treatments. Conducting complementary validations of the proposed approach is another direction for future work.

Reproducibility

The source code and documentation to reproduce the experiments are available at https://github.com/zahradbozorgi/CausalRulesDiscovery

Acknowledgments

Research funded by the Australian Research Council (grant DP180102839) and the European Research Council (PIX Project). Thanks to Tomáš Kliegr and Lukáš Sýkora for providing their Action Rules Jupyter notebook and to Mahmoud K. Shoush for his help with data preprocessing.

References

[1] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management, 2nd Ed.* Springer, 2018.
[2] T. Lehto, M. Hinkka, and J. Hollmén, "Focusing business improvements using process mining based influence analysis," in *Proc. of BPM*, ser. LNBIP, vol. 260. Springer, 2016.
[3] T. Lehto, M. Hinkka, and J. Hollmen, "Focusing business process lead time improvements using influence analysis," in *Proc. of SIMPDA)*, ser. CEUR Workshop Proceedings, vol. 2016. CEUR-WS.org, 2017.
[4] N. Gupta, K. Anand, and A. Sureka, "Pariket: Mining business process logs for root cause analysis of anomalous incidents," in *Proc. of DNIS Workshop*, ser. LNCS, vol. 8999. Springer, 2015.
[5] S. Suriadi, C. Ouyang, W. M. P. van der Aalst, and A. H. M. ter Hofstede, "Root cause analysis with enriched process logs," in *Proc. of BPM Workshops*, ser. LNBIP, vol. 132. Springer, 2012.
[6] B. F. A. Hompes, A. Maaradji, M. L. Rosa, M. Dumas, J. C. A. M. Buijs, and W. M. P. van der Aalst, "Discovering causal factors explaining business process performance variation," in *Proc. of CAiSE*, ser. LNCS, vol. 10253. Springer, 2017.
[7] 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 *Proc. of ICSOC*, ser. LNCS, vol. 9435. Springer, 2015.
[8] T. Narendra, P. Agarwal, M. Gupta, and S. Dechu, "Counterfactual reasoning for process optimization using structural causal models," in *Proc. of BPM Forum*, ser. LNBIP, vol. 360. Springer, 2019.
[9] A. Polyvyanyy, A. Pika, M. T. Wynn, and A. H. ter Hofstede, "A systematic approach for discovering causal dependencies between observations and incidents in the health and safety domain," *Safety Science*, vol. 118, 2019.
[10] W. M. P. van der Aalst, *Process Mining - Data Science in Action, Second Edition*. Springer, 2016.
[11] A. Dardzinska, *Action Rules Mining*, ser. Studies in Computational Intelligence. Springer, 2013, vol. 468.
[12] M. A. Hernan and J. M. Robins, *Causal Inference: What If*. Boca Raton: Chapman & Hall CRC, 2020.
[13] S. R. Künzel, J. S. Sekhon, P. J. Bickel, and B. Yu, "Metalearners for estimating heterogeneous treatment effects using machine learning," *Proc. of the National Academy of Sciences*, vol. 116, no. 10, 2019.
[14] P. Rzepakowski and S. Jaroszewicz, "Decision trees for uplift modeling with single and multiple treatments," *Knowl. Inf. Syst.*, vol. 32, no. 2, 2012.
[15] P. Gutierrez and J.-Y. Gérardy, "Causal inference and uplift modelling: A review of the literature," in *Proc. of Int. Conf. on Predictive Applications and APIs*, ser. Proc. of Machine Learning Research, vol. 67. PMLR, 2017.
[16] J. Pearl, *Causality*. Cambridge university press, 2009.
[17] Z. Zhao and T. Harinen, "Uplift modeling for multiple treatments with cost optimization," *CoRR*, vol. abs/1908.05372, 2019.
[18] H. Chen, T. Harinen, J.-Y. Lee, M. Yung, and Z. Zhao, "Causalml: Python package for causal machine learning," 2020.
[19] A. Rodrigues, C. Almeida, D. Saraiva, F. Moreira, G. Spyrides, G. Varela, G. Krieger, I. Peres, L. Dantas, M. Lana *et al.*, "Stairway to value: mining a loan application process," *BPI Challenge*, 2017.
[20] L. Blevi, L. Delporte, and J. Robbrecht, "Process mining on the loan application process of a dutch financial institute," *BPI Challenge*, 2017.
[21] E. Povalyaeva, I. Khamitov, and A. Fomenko, "Bpic 2017: density analysis of the interaction with clients," *BPI Challenge*, 2017.

Using Multi-Level Information in Hierarchical Process Mining: Balancing Behavioural Quality and Model Complexity

Sander J.J. Leemans
Queensland University of Technology Australia
s.j.j.leemans@qut.edu.au

Kanika Goel
Queensland University of Technology Australia

Sebastiaan J. van Zelst
Fraunhofer FIT & RWTH University Germany

***Abstract*—Process mining techniques aim to derive knowledge of the execution of processes, by means of automated analysis of behaviour recorded in event logs. A well-known challenge in process mining is to strike an adequate balance between the behavioural quality of a discovered model compared to the event log and the model's complexity as perceived by stakeholders. At the same time, events typically contain multiple attributes related to parts of the process at different levels of abstraction, which are often ignored by existing process mining techniques, resulting in either highly complex and/or incomprehensible process mining results. This paper addresses this problem by extending process mining to use event-level attributes readily available in event logs. We introduce (1) the concept of multi-level logs and generalise existing hierarchical process models, which support multiple modelling formalisms and notions of activities in a single model, (2) a framework, instantiation and implementation for process discovery of hierarchical models, and (3) a corresponding conformance checking technique. The resulting framework has been implemented as a plug-in of the open-source process mining framework ProM, and has been evaluated qualitatively and quantitatively using multiple real-life event logs.**

***Index Terms*—Process mining, process model complexity, process discovery, hierarchical process models, multi-level event logs**

I. Introduction

Process mining [1] techniques are concerned with deriving process-based insights from historical records of business operations recorded in information system event logs. Two often-used categories of process mining techniques are *process discovery* [5] and *conformance checking* [8]. Process discovery aims to extract understandable and representative process models from event logs in order for stakeholders to uncover the real behaviour of their business processes. Many process discovery algorithms exist; each focusing on generating process models with a certain level of quality, expressed using the dimensions of fitness, precision, simplicity and generalisation [1]. A well-known challenge for any process discovery technique is to strike the right balance between the behavioural quality of a discovered model with respect to the event log and the model's complexity as perceived by stakeholders. Applying most existing process discovery techniques on real-life event logs results either in detailed highly complex and incomprehensible models (due to high numbers of activities and complex behaviour) or abstract and simple but behaviourally inaccurate models (due to leaving out or adding behaviour with respect to the log).

Despite advances made in process discovery, the understandability of the discovered models is often compromised when dealing with complex processes. The reasons for obtaining such complex models are plentiful, e.g., noise is inherently present in logs. Furthermore, often, the abstraction level at which the event attributes are recorded is often much lower than which one ideally models a business process at [39]. At the same time, this problem can be alleviated by using *hierarchical process models* as a primary process representation [28]. A hierarchical process model is characterised by hierarchy of levels, each of which captures a particular granularity of the process. Then, each step in the process constitutes a step in potentially multiple levels of the hierarchy, which inherently leads to the notion of multi-level behaviour. For instance, the following *multi-level trace* represents a hospitalisation: the first four events denote a stay at the intensive care (level 1), during which the patient was intubated (level 2), which consisted of intubation, ventilation and extubation (level 3). Afterwards, the patient was deregistered from the IC and transfered to a ward.

$$\left\langle \begin{pmatrix} \text{IC} \\ \text{intubation} \\ \text{intubate} \end{pmatrix}, \begin{pmatrix} \text{IC} \\ \text{intubation} \\ \text{ventilate} \end{pmatrix}, \begin{pmatrix} \text{IC} \\ \text{intubation} \\ \text{extubate} \end{pmatrix}, \begin{pmatrix} \text{IC} \\ \text{transfer} \\ \text{deregister} \end{pmatrix}, \begin{pmatrix} \text{Ward 1} \\ \text{transfer} \\ \text{register} \end{pmatrix}, .. \right\rangle$$

Existing discovery techniques assume process steps are represented by a single attribute (e.g., `concept:name`), thus logs such as our example (e.g. [17]) would yield either high-level models containing lots of loops (level 1) or overly spaghetti-like complex models (level 3). As we will show in Section VI, real-life event logs readily contain many more such attributes. By considering multiple attributes, "hierarchical decomposition plays a central role for organising processes in an understandable way and for refining coarse-granular towards a fine-granular representation" [26]. Our evaluation illustrates this: Figure 4f illustrates that complex models can be made simpler by splitting them up in hierarchical parts.

The use of hierarchy to improve the understandability of process models is well established [3], [4], [28], [33], [36], and recent techniques have been proposed that abstract attributes in event logs: see [39] for a literature review.

978-1-7281-9832-3/20/$31.00 ©2020 IEEE
DOI 10.1109/ICPM49681.2020.00029

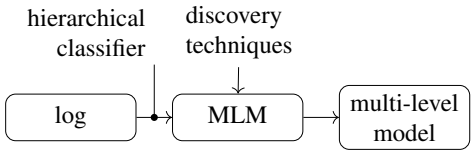


Figure 1: Context of the MLM framework.

In this paper, we introduce the concept of multi-level information in event logs, and propose hierarchical discovery and conformance checking techniques that can use this information. Existing hierarchical discovery techniques either do not consider multi-level information, are limited to 2 layers of hierarchy or are tied to specific combinations of model formalisms [25], [34], [36] (different types of behaviour may require different modelling concepts, e.g. declarative and imperative formalisms [12]). We introduce a framework for hierarchical process discovery that explicitly takes multi-level information into account and is limited in neither layers nor formalisms. This Multi-Level Miner (MLM) framework takes as input a standard event log and a multi-level classifier (which indicates the attributes involved and their order), then recursively abstracts groups of events, applies existing discovery techniques and splits the event log accordingly, to output a hierarchical process model (see Figure 1). *As such, MLM can be regarded as a standardisation effort for event logs that either inherently describe multi-level process information or that contain derived multi-level process information.*

We instantiate an algorithm using MLM and introduce a conformance checking technique to compare multi-level logs and hierarchical models. Then, we evaluate MLM with respect to existing discovery techniques, which shows that MLM can discover models with quality on par with existing techniques, while being arguably simpler to understand for users.

In short, the contributions of this paper are:

- A conceptual model and definition of multi-level logs, and definitions of generalised hierarchical models;
- An implemented, instantiated, extensible model discovery framework (MLM) that considers multi-level information;
- A corresponding conformance checking technique.

The remainder of the paper is organised as follows. Section II discusses related work. Section III introduces our multi-level languages and hierarchical process models, while Section IV presents how we discover them and Section V discusses how to check their conformance. Section VI presents key findings from our evaluations with multiple real-life event logs, and we discuss limitations in Section VII. Section VIII concludes the paper.

II. Related Work & Background

In this section, we discuss related work and concepts of process modelling formalisms, process discovery techniques, single- and multi-formalism hierarchical models, and techniques that identify multi-level structures in logs.

From an abstract point of view, a process model describes behaviour as a possibly infinite set of traces, a *language*, over the *transitions* present in the model. For instance, directly follows models/transition systems use arcs to express which transitions can be executed after each transition [24]. Process trees are a hierarchy of nodes (with transitions as leaves), each combining the languages of its sub-nodes [20]. BPMN models express complex behaviour using advanced gateways and exception handling events [9]. Declarative models explicitly express constraints over pairs of transitions [35]. Finally, a Petri net describes behaviour through occurrences of transitions that consume tokens from states (places) and produce tokens on places [29]. We assume the reader to be familiar with the basic principles of the most commonly used process mining formalisms; see [1] for an introduction.

Process modelling formalisms that support hierarchy include process trees [20], BPMN [9], declarative languages [33] and fuzzy models [15]. Our definition (Section III) generalises over these formalisms by providing a multi-formalism hierarchy, and establishes their multi-level semantics.

A plethora of non-hierarchical discovery techniques has been proposed. For a detailed overview of these techniques, please refer to [1], [5]. The approach of this paper can use any discovery technique to improve on the simplicity and understandability of the discovered models at any level, by using multi-level information from the event log.

Techniques that identify or create levels in event logs include clustering activities based on correlations [16], identifying stages [30], [9], or constructing taxonomies [14]. For a recent literature review, see [39]. Using these techniques, multi-level information can be added to event logs, which can be used as input (levels) for the MLM framework. No technique that explicitly uses this information has been proposed.

Next, we discuss approaches that aim to discover hierarchical models. In [20], process trees are discovered by splitting the event log recursively into smaller parts; the hierarchy bears no meaning (as in [37]). This is combined with other techniques in [22], where a hierarchy is obtained by combining discovery techniques: where one discovery technique gives up, another is started in a hierarchical way. In [19], hierarchy is used to express recursion, derived from inherently hierarchical software stack traces, using process trees. In [9], [10], first events are clustered, after which for each cluster BPMN models are discovered in a hierarchical way. Similarly, in [33], text mining is applied to identify related activity labels, which are consecutively collapsed in the event log and used to discover a hierarchical declarative process model. Finally, the Fuzzy Miner [15] dynamically groups activities based on their already-discovered relation with other activities, and thereby implicitly induces a hierarchy.

None of the aforementioned techniques explicitly takes multi-level information into account, i.e., even though the techniques allow us to derive a model comprising a hierarchy, the primary input is still of a "flat" nature. Moreover, none of the techniques is able to combine multiple process modelling formalisms, i.e., allow us to pick the most suitable modelling formalism for a specific level within the process.

Hybrid models combine multiple modelling formalisms to reduce complexity, in particular they use declarative and

imperative parts. In hybrid models, a transition can contain a sub-model of a different formalism. To define the semantics of hybrid models, the open-world assumption, trace termination and concurrency in atomic languages (such as Petri net) need particular attention [35]. While in hybrid models only a lowest-level transition generates an event, the formalism introduced in this paper generalises over this approach by adding support for multi-level events. Discovery techniques for hybrid models were proposed in [25], [34], [36]. Neither of these approaches consider multi-level information in event logs and the last two are limited to 2 layers, as they specifically aim to combine declarative models as sub-models of imperative models. This paper generalises these techniques by combining multiple formalisms without limiting the number of layers. Furthermore, we specifically use multi-level information in the event log to control the creation of hierarchical levels whereas previous works derive this information using e.g. clustering.

Finally, artefact-centric process mining considers the relation between multiple process/case notions [31], [13], such as orders, products and packages. This notion differs from multi-level as we assume that one level is fully contained in another, where artefact-centric assumes no such relation. Similarly, multi-dimensional process mining [38] does not consider hierarchies in a single process model.

III. Multi-Level Event Logs and Hierarchical Models

In an information system, an event e represents the occurrence of an activity. We generalise this to each event having several activities, at different levels of abstraction.

Definition 1 (Multi-level language, log)**.** *Let $\Sigma_1 ... \Sigma_m$ be alphabets of activities. Then, a* multi-level event *is a vector $(a_1, \ldots, a_m)$ with $\forall_{1 \leqslant i \leqslant m} a_i \in \Sigma_i$ of values describing a process step. A* multi-level trace *is a finite sequence of multi-level events, a* multi-level language *is a collection of multi-level traces and a* multi-level log *is a multi-level language with a finite number of traces.*

An example of a multi-level event is (Intensive Care, Intubation, Extubate), which indicates that an activity "extubate" (removing an air pipe from a patient's throat) was performed, which is the final stage of a higher-level activity/procedure "intubation", which in turn is a step in an intensive care stay. A multi-level trace in this example would be a patient being admitted to the intensive care, and the corresponding multi-level event log would be all intensive care stays recorded.

Every (standard) event log that contains multiple event attributes can be transformed into a multi-level log by deciding on the attributes that indicate the levels, and their order, using a *multi-level classifier*, which is a list of classifiers, which are lists of attributes (i.e. columns in a CSV file).

A *hierarchical model* combines process models of several formalisms to hierarchically express a multi-level language. We define hierarchical models recursively using a process model M of any formalism:

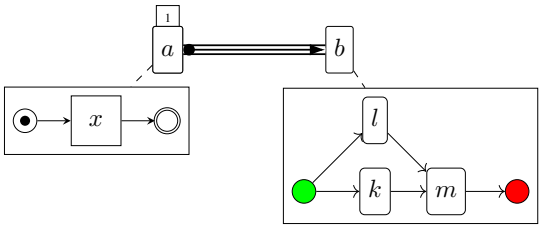


Figure 2: A multi-level process model, consisting of a Declare model (top), a Petri net and a Directly Follows model.

Definition 2 (Hierarchical model - syntax)**.** *Let Σ be an alphabet of activities. Then, a* hierarchical model *M is a process model $M(T_1, \ldots T_n)$ expressed over transitions $T_1 \ldots T_n$, where a* transition *T_i is either: (1) an activity $a \in \Sigma$; (2) a silent step $\tau \notin \Sigma$; or (3) an annotated activity $a_{M'}$ in which $a \in \Sigma$ and M' is a hierarchical model.*

Figure 2 shows an example, consisting of a Declare model, of which activity a is annotated with a sub-model (a Petri net), and activity b is annotated with a directly follows model.

In the following definition, Y replaces a single execution of a transition with the sub-model language of the transition, while X exhaustively calls Y until all transitions have been replaced:

Definition 3 (Hierarchical model - semantics)**.** *Let M be a model over transitions $T_1 \ldots T_n$. Let $\varphi(M(T_1, \ldots T_n))$ be a function that returns the language of M in terms of starts and completes of its transitions, that is, over $T_{1,s}, T_{1,c}, \ldots T_{n,s}, T_{n,c}$. Then, the multi-level language of M is:*

$$
\begin{aligned}
\mathcal{L}(M) &= X(\varphi(M)) \\
X(L) &= \begin{cases} X(Y(L)) & \text{if } \exists_{t \in L} \exists_i T_{i,s} \in t \\ L & \text{otherwise} \end{cases} \\
Y(L) &= \{t' \mid t \in L \\
&\quad \wedge t = t_1 \cdot \langle T_{i,s} \rangle \cdot t_2 \cdot \langle T_{i,c} \rangle \cdot t_3 \wedge T_{i,c} \notin t_2 \\
&\quad \wedge t' \in \{t_1\} \bullet (\mathcal{L}(T_i) \parallel \{t_2\}) \bullet \{t_3\}\} \\
&\quad \cup \{t' \in L \mid \neg \exists_{t \in L} \exists_i T_{i,s} \in t\} \\
\mathcal{L}(a) &= \{\langle (a) \rangle\} \\
\mathcal{L}(\tau) &= \{\langle\rangle\} \\
\mathcal{L}(a_{M'}) &= \{\langle \binom{a}{e_1}, \ldots \binom{a}{e_m} \rangle \mid \langle e_1, \ldots e_m \rangle \in \mathcal{L}(M')\}
\end{aligned}
$$

Where $\bullet$ is the sequential cross product and $\parallel$ is the interleaving cross product [21].

For instance, the multi-level language of the model shown in Figure 1 is $\{\langle \binom{a}{x}, \binom{b}{l}, \binom{b}{m} \rangle, \langle \binom{a}{x}, \binom{b}{k}, \binom{b}{m} \rangle\}$.

Hierarchical models are defined for any formalism for which φ can be defined, which entails the introduction of true concurrency between transition executions (which can be simulated in atomic Petri nets, process trees and directly follow models; see Section V). The only requirement to this φ is that for every trace generated, there is a bijective mapping of every

$T_{i,s}$ to/from a later following $T_{i,c}$. However, for declarative models, other factors (e.g. the open-world assumption) need to be considered [35].

In the remainder of this paper, for simplicity, we assume that the hierarchy of a hierarchical model is of a consistent depth, even though this is not required by our definitions (that is, such models simply emit multi-level events of uneven sizes), and we may omit the "multi-level" prefix where appropriate.

IV. Discovering Multi-Level Models

In this section, we introduce the *Multi-Level Miner (MLM)* framework. We first give its pseudocode and explain its key steps, after which we give a specific algorithm implementing it, and we sketch strategies in case extra information is available.

A. The Multi-Level Miner Framework

As input, MLM takes a multi-level language (which can be obtained from an event log using a multi-level classifier) and a list of discovery techniques $D_1 \dots D_m$ to be applied over m levels of attributes.

Intuitively, MLM first prepares the event log by abstracting groups of events to the current level l, then applies a discovery technique, and finally recurses on the transitions of the discovered model, replacing each transition with a process model of the next level. Thus, the framework uses two key functions that algorithms implementing the framework must provide for each level $l \in \{1 \dots m\}$: event group abstraction P_l and log splitting S_l. The event group abstraction function P_l aims to remove repetition from traces. For instance, a "flat" discovery technique would falsely consider the trace $\langle a, b, b \rangle$ to contain a repetition of b. However, these bs indicate lower level steps, rather than a repetition of b itself. Thus, P_l abstracts this trace into $\langle a, b \rangle$ by removing the repeated b. After a process discovery technique has been applied, the log splitting step S_l splits the (non-abstracted) log into sub-logs: one sub-log for each transition in the discovered process model. Intuitively, each sub-log contains a trace for each execution of the corresponding activity. In our example, the trace would be split into $\langle a \rangle$ and $\langle b, b \rangle$.

To formally define MLM, we use a projection function ϕ that projects a particular level of an event to its l^{th} attribute value, and by extension multi-level traces and multi-level languages onto traces and languages: $\phi((a_1, \dots a_m), j) = a_j$. Then, the MLM framework is defined as follows:

procedure $MLM_{P,D,S}$(multi-level language L, level l)
  if l is the bottom level **then**
    $M \leftarrow D_l(\phi(L, l))$ ▷ discover a process model
  else
    $L' \leftarrow P_l(L)$ ▷ abstract groups in L
    $M \leftarrow D_l(\phi(L', l))$ ▷ discover a process model
    for all transition $t \in M$ **do**
      $L_t \leftarrow S_l(M, L)$ ▷ split L using M
      $M_t \leftarrow$ MLM($L_t, l{+}1$) ▷ recurse
      replace t with t_{M_t} in M ▷ link t to sub-model M_t
    end for
  end if
  return M
end procedure

B. An Example Instantiation

To illustrate the MLM framework, we instantiate it in an actual algorithm as follows. For the log abstraction function P_l, consecutive appearances of the same activity are removed. For instance, the trace $\langle \binom{a}{x}, \binom{a}{y}, \binom{b}{x} \rangle$ at level 1 would be filtered to $\langle \binom{a}{x}, \binom{b}{x} \rangle$. For the log splitting function S_l, (1) the discovered model is transformed to allow for repeated executions of activities by replacing each transition as in Table I; (2) alignments [2] are computed to link events in the log to a transition in the transformed model, and to remove non-fitting events, which makes the instantiation non-deterministic; and (3) for each transition T, the trace is split into sub-trace(s): events that are not linked to T are removed, and a new sub-trace is started depending on the model formalism. [directly follows models] a new sub-trace is started when an event occurs of a transition $T' \neq T$ (as directly-follows models do not support concurrency); [process trees] a new sub-trace is started when an event occurs of a transition $T' \neq T$ such that T' is not concurrent with T in the tree; [Petri nets] a new sub-trace is started when an event occurs of any transition.

For instance, consider the top-level model of Figure 2 and the trace $\langle \binom{a}{x}, \binom{b}{k}, \binom{b}{m} \rangle$ at level 1. This trace would be split into the subtrace $\langle \binom{a}{x} \rangle$ for transition a and the subtrace $\langle \binom{b}{k}, \binom{b}{m} \rangle$ for transition b.

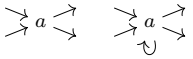
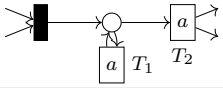
V. Conformance Checking

To compare a hierarchical model with a multi-level log, existing state-space exploring conformance checking techniques could be updated to support multi-level logs. However, we can use existing techniques unchanged by translating a hierarchical model to a workflow net (*flattening*), if all levels are individually translatable to workflow nets. In the following, let M be a process model translated to a workflow net over transitions $(T_1, \dots T_n)$, which are replaced as follows:

a, τ are not replaced;

$a_{M'}$ (1) the sub-model M' is recursively translated to a workflow net having a source place p_{source} and a sink place p_{sink}; (2) all labelled transitions of M' get their labels superpositioned with a, e.g. $(x) \rightarrow \binom{a}{x}$; (3) a silent transition t_{start} is added, which has the same input arcs as $a_{M'}$ and one output arc to p_{source}; and (4) another silent transition t_{end} is added with one input arc from p_{sink} and

Table I: Log splitting: model transformation for several formalisms.

Formalism	transition	transformed
Directly follows model		
Process tree	a	$\circlearrowleft(a, \tau)$
Petri net		

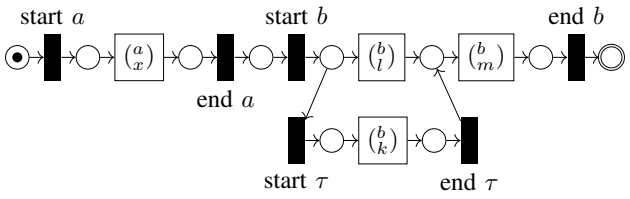


Figure 3: Our multi-level model flattened into a Petri net.

the same output arcs as $a_{M'}$. The remaining net is sound, that is, free of deadlocks and other anomalies [20], if all of the sub-model-workflow nets are sound;

For instance, the hierarchical model shown in Figure 2 would be flattened to Figure 3.

VI. Evaluation

In this section, we evaluate the MLM framework: first, we illustrate its applicability in a qualitative comparison with other discovery techniques. Second, we evaluate the quality of the models discovered by the MLM framework.

Reproducibility. All event logs used in this paper are publicly available, as well as the source code of our technique.

A. Implementation

Both the MLM framework instantiation (Section IV-B) and the conformance checking technique (Section V) have been implemented as plug-ins of the ProM framework [11], in package `multi-level miner`. The MLM and its implementation allow for easy extension with other discovery techniques and sub-model formalisms; currently Petri nets, directly follows models, BPMN models and process trees are supported. The resulting hierarchical model can be visualised in two ways: either as a single model (as in Figure 4f), or using a separate visualisation for each level, where a user can click on an annotated node, which opens the corresponding sub-model (Figure 4e).

B. Qualitative Evaluation

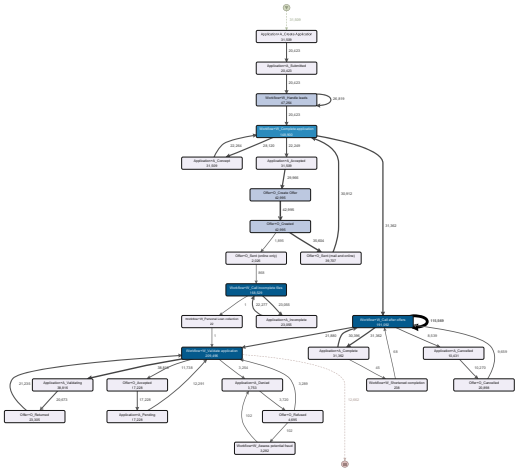
To study qualitative and understandability aspects of the MLM framework, we applied Disco (see http://fluxicon.com), Declare [35], Inductive Miner - infrequent [20] and Split Miner [6] at their default settings to the BPI Challenge 2017 log, which was recorded from a loan application process at a financial institution in the Netherlands. Based on domain knowledge and a small explorative analysis, we chose the attributes `eventOrigin` and `concept:name` as the multi-level classifier, and chose the Directly Follows Miner [24] and the Inductive Miner as discovery techniques for the MLM framework. The discovered models are shown in Figure 4. A visual inspection of these models immediately reveals that the multi-level model is much simpler and understandable than any of the other models, especially on more abstract levels. Hierarchical models enable an understanding of the information in the logs in a systematic manner, by displaying the behaviour in layers. For instance, the model obtained using Disco may not be complex on first sight (even though it is highly filtered as per default), although, understanding the flow of activities under a particular event origin is not easy, unlike in multi-level models. Other models might be more structured (Inductive Miner), more flexible (Declare) or free of concurrency (Split Miner), however the MLM model is arguably simpler.

In [28], seven Process Modelling Guidelines for understandability were identified, of which four are relevant for this experiment: (G1) use as few model elements as possible: compared to the other process models, the MLM model decomposes information in layers which contain fewer elements. This enables users to focus on a particular section of the model rather than on all the elements at the same time, even though the total number of elements might be higher. (G2) minimise the routing paths per element: the experiment did not show large differences between the techniques. (G4) model as structured as possible. MLM introduces structure by its layers, which for some models increases the structuredness of the entire model compared to other techniques. (G7) decompose a model with more than 30 elements: our technique enables decomposition of the model increasing the understandability of the discovered model. Overall, we can see that our model improves on most of the relevant guidelines, enabling better understanding of the behaviour in the logs.

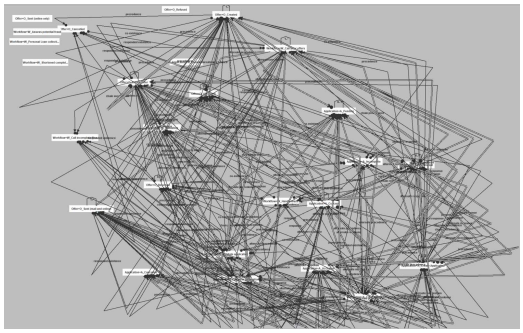
C. Quantitative Evaluation

To compare the MLM framework with existing discovery techniques, we applied several techniques to six publicly available real-life event logs. These logs are publicly available from https://data.4tu.nl/repository/collection:event_logs_real, and were chosen as they contain multiple event attributes that might serve as activities. As a first step, for each log we manually chose a multi-level classifier, based on domain knowledge on the mentioned web page. The event logs are summarised in Table II, which also shows the chosen classifiers. The included existing discovery techniques are Split Miner (SM) [6], Inductive Miner - infrequent (IM) [20] and Directly Follows Model Miner (DM) [24]). Furthermore, we included two baseline techniques: the trace model (T) that represents all traces of the event log, and the flower model (F) that represents all possible behaviour given the activities in the log. Finally, we included several instantiations of the MLM framework: (a) three instantiations that use the same existing discovery technique for each of its levels, using Split Miner (MLM-SM), Inductive Miner - infrequent (MLM-IM) and Directly Follows Model Miner (MLM-DM); (b) an instantiation that uses the flower model for each of its levels (MLM-F); a trace-MLM would not involve multiple layers and was not included; (c) an instantiation that combines Directly Follows Model Miner and Inductive Miner - infrequent (MLM-DM-IM).

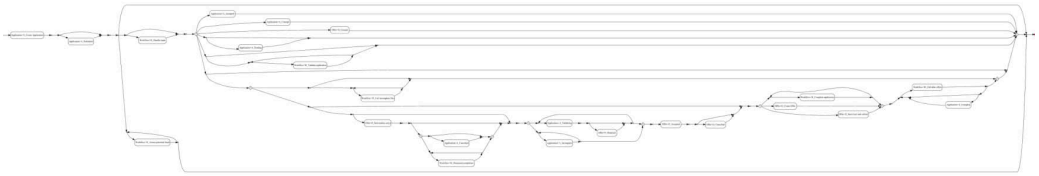
To these logs and discovery techniques, we applied 3-fold cross validation: the log was split into 3 buckets, and for each bucket a model was discovered, which was validated against the remaining buckets. To minimise the influence of randomness in the cross validation, this validation was repeated 10 times. Thus, for each log and discovery technique, 30 models



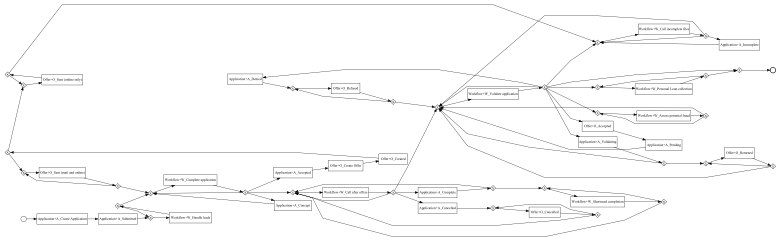
(a) Directly follows model (Fluxicon Disco).



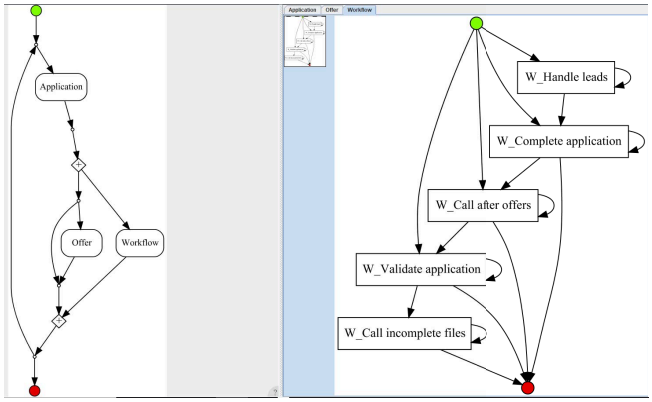
(b) Declarative model (Declare).



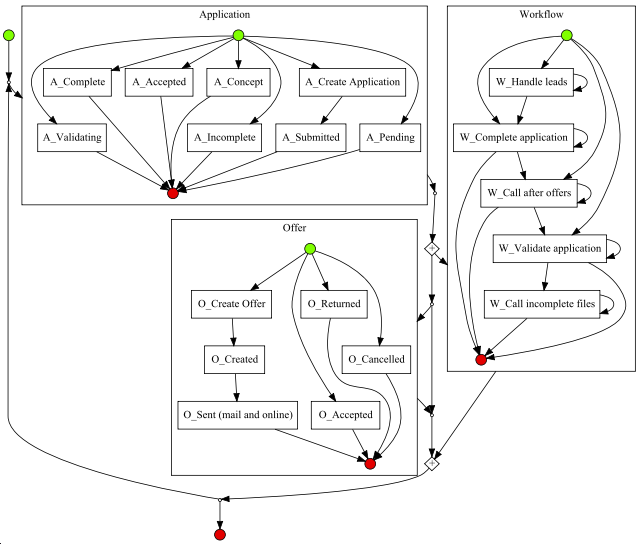
(c) Process tree (Inductive Miner).



(d) BPMN model (Split Miner).



(e) This paper - MLM-DM-IM (tabbed).



(f) This paper - MLM-DM-IM (hierarchical).

Figure 4: Impressions of mined process models for our qualitative evaluation.

Table II: Event logs used in the quantitative evaluation.

log	multi-level classifier	traces	events	acts
bpic 2012	concept:name, lifecycle:transition	13087	262200	36
bpic 2013 closed problems	concept:name, lifecycle:transition	1487	6660	7
bpic 2015 - 1	monitoringResource, activityNameEN	1199	52217	2302
bpic 2017	eventOrigin, concept:name	31509	1202267	26
bpic 2018 payment applications	subprocess, concept:name, lifecyle:transition	43809	984613	52
sepsis	org:group, concept:name	1050	15214	42

were discovered and evaluated. The reported measures are the averages over these 30 measures.

For each log and discovered model, we measured projected fitness and precision [23] on a Petri-net representation of the model; the MLM-models were flattened using the method described in Section V. Furthermore, we recorded the total number of nodes and edges, and the average connector degree. Together, these measures have been shown to be significantly negatively correlated with understandability [27], [32]. Both the total number of nodes and edges and the average connector degree were measured on a flattened Petri net (as a machine would) and on the native graph visualisation of the model (as a user would), where for MLM-models we averaged the values over the sub-models in the MLM-models. We refer to these measures as automated size and user size.

Table III shows an excerpt of the results, with the full results available in a technical report at https://eprints.qut.edu.au/134958/. While run time of MLM was considerably longer than for the other discovery techniques, due to the repeated application of alignments, in this experiment run time never exceeded two hours, and was generally lower than 20 minutes. Thus, while feasible for the real-life logs considered here, larger or more complex logs with more traces, events per trace or activity might prove challenging. For some models, fitness and precision could not be measured. Remarkably, on bpic17, several variants of MLM got the same precision as the flower model, even though the corresponding models (Figure 4f) clearly limit behaviour considerably, and this was not the case for the other logs. Comparing an algorithm and the same algorithm in MLM, we see slight differences in both directions for fitness and precision, but consistently smaller models. For all event logs, F was pareto optimal (over fitness, precision and user-average-size) once, T 5 times, DF 4 times, IM 3 times, SM 3 times, MLM-F 5 times, MLM-DM 5 times, MLM-IM once, MLM-SM once, and MLM-DM-IM 4 times. MLM techniques got the highest fitness for 3 logs (disregarding the baselines MLM-F and F), and the highest precision for 2. This shows that MLM scores comparable or better on the key metrics that are related to the understandability of process models, and comparable on fitness and precision, compared to existing techniques.

We conclude that even though the hierarchical models are slightly more complex for automated analysis (as elements are introduced with flattening), these models are much simpler for user analysis (considering the average size per level/sub-model), and have a similar quality as existing techniques.

VII. Discussion & Limitations

While the positive effects of hierarchy on understandability are well-established in process modelling literature [3], [4], [28], [33], [36], the extent to which multiple formalisms might increase complexity requires further research. Thus, more research is necessary into simplicity measures that compare hierarchical and "flat" process models fairly, and into users' perceptions of these measures.

In our evaluation, we selected multi-level classifiers based on domain knowledge and on a quick analysis of short loops for each attribute. We argue that this adds little complexity over manually choosing a single classifier, as it requires choosing multiple columns of a CSV file rather than a single one. Obviously, the outcomes of multi-level process mining highly depend on this choice, it may introduce a bias of the results, and it requires that multiple suitable attributes are present in the log. Users might need to iterate to find suitable multi-level classifiers, discovery techniques and formalisms. In future work, it would be interesting to explore methodologies, scoring and best-practices to recommend these (e.g. as in [7]), and to study their influence in detail.

The abstraction step (5) of MLM suffers from a chicken-and-egg problem when regarding concurrency: in order to abstract, knowledge of the process model would be beneficial, however that is not available at that step. This manifests in our instantiation in two situations. First, our event-group abstraction step cannot be aware of concurrency: if executions of two activities a and b overlap by having alternating events, then our instantiation will split these executions whenever an alternating event occurs. For instance, $\langle \binom{a}{x}, \binom{b}{y}, \binom{a}{y}, \binom{b}{x} \rangle$ would be split into $\langle \binom{a}{x} \rangle$ and $\langle \binom{a}{y} \rangle$ for a, and $\langle \binom{b}{y} \rangle$ and $\langle \binom{b}{x} \rangle$ for b, rather than $\langle \binom{a}{x}, \binom{a}{y} \rangle$ and $\langle \binom{b}{y}, \binom{b}{x} \rangle$. However, if the discovery technique (step 6) nevertheless discovers the concurrency, the log splitting (step 8) will split the log correctly. Second, repeated execution of an activity (length-1-loops) challenges both abstraction and log splitting: given a repeated execution of an activity (for instance, $\langle a, a, a, a \rangle$), without further information, both steps cannot decide how many executions of a are present, nor where one execution ends and the next one starts. In future work, both could be addressed using attributes such as `concept:instance` or `lifecycle:transition`.

Finally, our conformance checking approach could be extended to support declarative formalisms [35].

VIII. Conclusion

Process mining is challenged by the need to strike a good balance between model complexity and behavioural quality of process models. In this paper, we introduced multi-level logs and generalised hierarchical models, which enable the use of multi-level data in existing logs to discover simpler models, and generalise over existing approaches by combining multiple formalisms with multiple levels of abstraction. We introduced a corresponding discovery framework and conformance checking technique, working on standard event logs,

Table III: Quantitative experiment results, with pareto-optimal results in **bold**. For all 6 logs, see https://eprints.qut.edu.au/134958/.

log	algorithm	fitness	precision	automated		user		time
				size	avg con deg	avg size	avg conn deg	(ms)
bpic17	F	1.00	0.60	78.50	3.85	58.67	1.97	1175.00
	T	**1.00**	**0.73**	2121196.70	2.02	**1088537.93**	2.00	816.97
	DM	**0.99**	**0.71**	213.23	2.90	**213.23**	2.90	4605.50
	SM	0.91	0.69	218.00	2.54	142.00	2.90	43399.23
	IM	0.95	0.69	178.77	2.67	147.13	1.99	370274.87
	MLM-F	**1.00**	**0.60**	111.50	3.33	**34.63**	2.83	1115102.67
	MLM-DM	**0.95**	**0.69**	237.00	3.10	**10.00**	1.06	31809.00
	MLM-IM	1.00	0.60	214.17	2.73	46.10	1.95	147507.60
⋮	MLM-SM	1.00	0.60	235.67	2.73	45.08	2.80	68784.10
	MLM-DM-IM	1.00	0.60	437.80	2.84	42.10	1.97	136822.30

where a user selects several attributes. The proposed approach has been evaluated in two ways: a qualitative evaluation on a real-life event log demonstrated that discovered hierarchical models can be much simpler and more understandable than flat models. Second, a quantitative evaluation on six real-life event logs showed that our approach scores better on the key metrics that are related to the understandability of process models, and comparable on fitness and precision to existing techniques.

There are several avenues for future work. Better ways to visualise hierarchical process models and evaluate the impact of visualisation on the understandability of these models could be explored. Also, automatic suggestions for suitable attributes and corresponding discovery techniques could be developed [7], and our approach could be combined with clustering-based hierarchical discovery techniques [39], [10]. Furthermore, performance measures for hierarchical models can be investigated, similar to [18].

Acknowledgement. We thank Dirk Fahland and Moe Wynn for their contributions to earlier versions of this paper.

REFERENCES

[1] van der Aalst, W.M.P.: Process Mining - Data Science in Action. Springer (2016)
[2] van der Aalst, W.M.P., et al.: Replaying history on process models for conformance checking and performance analysis. DMKD **2**(2), 182–192 (2012)
[3] Andaloussi, A.A., et al.: Evaluating the understandability of hybrid process model representations using eye tracking: First insights. In: BPM workshops. pp. 475–481 (2018)
[4] Andaloussi, A.A., et al.: Exploring the understandability of a hybrid process design artifact based on DCR graphs. In: CAiSE workshops. pp. 69–84 (2019)
[5] Augusto, A., et al.: Automated discovery of process models from event logs: Review and benchmark. TKDE **31**(4), 686–705 (2019)
[6] Augusto, A., et al.: Split miner: automated discovery of accurate and simple business process models from event logs. KaIS **59**(2), 251–284 (2019)
[7] Back, C.O., et al.: Towards an empirical evaluation of imperative and declarative process mining. In: ER workshops. pp. 191–198 (2018)
[8] Carmona, J., et al.: Conformance Checking - Relating Processes and Models. Springer (2018)
[9] Conforti, R., et al.: Beyond tasks and gateways: Discovering bpmn models with subprocesses, boundary events and activity markers. In: BPM. pp. 101–117 (2014)
[10] Conforti, R., et al.: BPMN miner: Automated discovery of BPMN process models with hierarchical structure. IS **56**, 284–303 (2016)
[11] van Dongen, B.F., et al.: The ProM framework: A new era in process mining tool support. In: Petri Nets. pp. 444–454 (2005)
[12] Fahland, D., et al.: Declarative versus imperative process modeling languages: The issue of understandability. In: CAiSE workshops. pp. 353–366 (2009)
[13] Fahland, D., et al.: Conformance checking of interacting processes with overlapping instances. In: BPM. pp. 345–361 (2011)
[14] Greco, G., et al.: Mining taxonomies of process models. DKE **67**(1), 74–102 (2008)
[15] Günther, C.W., van der Aalst, W.M.: Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In: BPM. pp. 328–343 (2007)
[16] Günther, C.W., Rozinat, A., van der Aalst, W.M.: Activity mining by global trace segmentation. In: BPM. pp. 128–139 (2009)
[17] Johnson, A.E., et al.: Mimic-iii, a freely accessible critical care database. Scientific data **3**, 160035 (2016)
[18] Leemans, M., et al.: Hierarchical performance analysis for process mining. In: ICSSP. pp. 96–105 (2018)
[19] Leemans, M., et al.: Recursion aware modeling and discovery for hierarchical software event log analysis. In: SANER. pp. 185–196 (2018)
[20] Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: BPM workshops. pp. 66–78 (2013)
[21] Leemans, S.J.J., et al.: Discovering block-structured process models from event logs - A constructive approach. In: Petri Nets. pp. 311–329 (2013)
[22] Leemans, S.J.J., et al.: Indulpet miner: Combining discovery algorithms. In: CoopIS. vol. 11229, pp. 97–115 (2018)
[23] Leemans, S.J.J., et al.: Scalable process discovery and conformance checking. SoSyM **17**(2), 599–631 (2018)
[24] Leemans, S.J.J., et al.: Directly follows-based process mining: Exploration & a case study. In: ICPM. pp. 25–32 (2019)
[25] Maggi, F.M., et al.: The automated discovery of hybrid processes. In: BPM. pp. 392–399 (2014)
[26] Malinova, M., et al.: An empirical investigation on the design of process architectures. In: ICW. pp. 1197–1211 (2013)
[27] Mendling, J., et al.: What makes process models understandable? In: BPM. pp. 48–63 (2007)
[28] Mendling, J., et al.: Seven process modeling guidelines (7PMG). IST **52**(2), 127–136 (2010)
[29] Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77**(4), 541–580 (1989)
[30] Nguyen, H., et al.: Stage-based discovery of business process models from event logs. IS **84**, 214–237 (2019)
[31] Popova, V., et al.: Artifact lifecycle discovery. IJCIS **24**(01), 1550001 (2015)
[32] Reijers, H.A., Mendling, J.: A study into the factors that influence the understandability of business process models. TSMCP **41**(3), 449–462 (2011)
[33] Richetti, P.H.P., et al.: Declarative process mining: Reducing discovered models complexity by pre-processing event logs. In: BPM. vol. 8659, pp. 400–407 (2014)
[34] Schunselaar, D.M.M., et al.: Mining hybrid business process models: A quest for better precision. In: BIS. pp. 190–205 (2018)
[35] Slaats, T., et al.: The semantics of hybrid process models. In: CoopIS. pp. 531–551 (2016)
[36] Smedt, J.D., et al.: Fusion miner: Process discovery for mixed-paradigm models. DSS **77**, 123–136 (2015)
[37] Verbeek, E.: Decomposed process mining with divideandconquer. In: BPM (Demos). p. 86 (2014)
[38] Vogelgesang, T., et al.: Multidimensional process mining: Questions, requirements, and limitations. In: CAiSE. vol. 1612, pp. 169–176 (2016)
[39] van Zelst, S., et al.: Event abstraction in process mining -literature review and taxonomy. Granular Computing (04 2020)

Discovering Hierarchical Processes Using Flexible Activity Trees for Event Abstraction

Xixi Lu*, Avigdor Gal†, and Hajo A. Reijers*
*Dept. of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
x.lu@uu.nl, h.a.reijers@uu.nl
†Faculty of Industrial Engineering and Management
Technion – Israel Institute of Technology, Haifa, Israel
avigal@technion.ac.il

***Abstract*—Processes, such as patient pathways, can be very complex, comprising of hundreds of activities and dozens of interleaved subprocesses. While existing process discovery algorithms have proven to construct models of high quality on clean logs of structured processes, it still remains a challenge when the algorithms are being applied to logs of complex processes. The creation of a multi-level, hierarchical representation of a process can help to manage this complexity. However, current approaches that pursue this idea suffer from a variety of weaknesses. In particular, they do not deal well with interleaving subprocesses. In this paper, we propose FlexHMiner, a three-step approach to discover processes with multi-level interleaved subprocesses. We implemented FlexHMiner in the open source Process Mining toolkit ProM. We used seven real-life logs to compare the qualities of hierarchical models discovered using domain knowledge, random clustering, and flat approaches. Our results indicate that the hierarchical process models that the FlexHMiner generates compare favorably to approaches that do not exploit hierarchy.**

***Index Terms*—Automated Process Discovery; Process Mining; Event Abstraction; Model Abstraction; Hierarchical Process Discovery**

I. Introduction

Complex processes often comprise of hundreds of activities and dozens of subprocesses that run in parallel [1], [2]. For example, in healthcare, a patient follows a certain process that consists of several procedures (e.g., lab test and surgery) for treating a medical condition. Studies have shown qualitatively that using hierarchical process models to represent complex processes can improve understandability and simplicity by "hiding less relevant information" [3]. In particular, the use of modularized, hierarchical process models, where process activities are collected into subprocesses, can be presented in a condensed higher-level presentation, revealing more details upon request.

Process discovery, a prominent task of process mining, aims at automatically constructing a process model from an event log. Over the years, dozens of discovery algorithms have been proposed [4]–[7]. These have proven to perform very well on relatively clean logs of structured processes. However, given the complexity of real-life processes, these algorithms tend to discover overfitted or overgeneralized models that are difficult to comprehend [8]. Existing discovery algorithms tend to disregard any hierarchical decomposition, whereas discovering hierarchical process models may help decrease the complexity of the discovery task.

Few hierarchical process discovery algorithms have been proposed [9]–[14]. While some approaches, such as [12], aim to automatically detect subprocesses, these algorithms are rigid in their assumptions and require extensive knowledge encoding for every event in the log regarding the causalities between the activities. Other approaches assume that subprocesses do not interleave [9], [14], which lead to discovering inaccurately, overly segmented models (see Fig. 1). It is also unclear how these approaches compare to conventional, flat models in terms of quality measures such as fitness and precision.

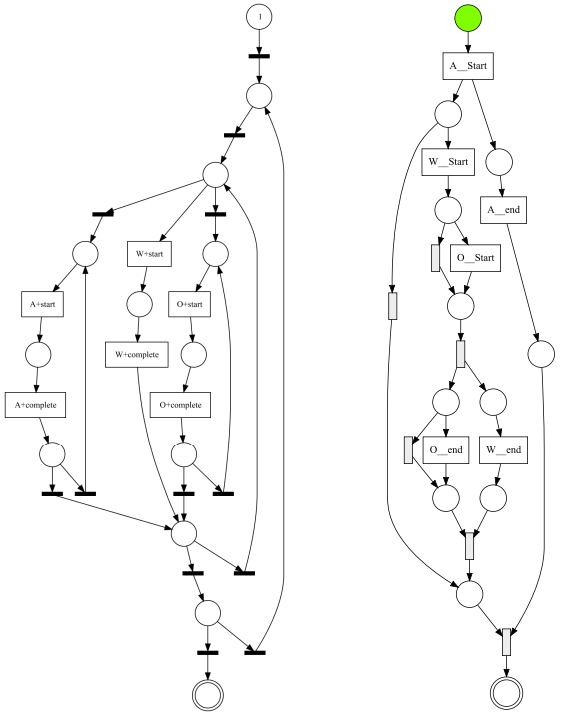
In this work, we propose FlexHMiner (FH), a three-step approach for the discovery of hierarchal models. We formalize the concept of *activity tree* and *event abstraction*, which allows us to be flexible in the ways of computing the process hierarchy. We illustrate this flexibility by proposing three different techniques to discover an activity tree: (1) a fully domain-based approach (DK-FH), (2) a random approach (RC-FH), and (3) a fall-back, flat activity tree (F-FH). After obtaining an activity tree, the second step of our approach is to compute the logs for each subprocess using log abstraction and log projection. Finally, FlexHMiner discovers a subprocess model for each subprocess by leveraging the capabilities of existing discovery algorithms. Using the domain-based approach as the gold standard and the flat tree approach as base line, we compare the three ways of discovering an activity tree using seven real-life logs.

The main contribution of this work is a novel approach to discover hierarchical process models from event logs, which (1) can handle multi-level interleaving subprocesses and (2) is flexible in its data requirements so that it can adapt to the situations when there is domain knowledge and when there is not. We evaluated FlexHMiner using seven real-life, benchmark event logs[1].

In the remainder, we define the research problem in Sect. II. The proposed approach is described in Sect. III. The evaluation

[1]The source code and the results can be found at: github.com/xxlu/prom-FlexHMiner

978-1-7281-9832-3/20/$31.00 ©2020 IEEE
DOI 10.1109/ICPM49681.2020.00030



(a) The sequential subprocessses discovered by SCM [14].

(b) The interleaving subprocesses discovered by our DK-FH.

Fig. 1: Difference in the discovered root models for the BPIC2012 log.

TABLE I: An example of an event log of a healthcare process.

$\mathcal{E}$	Patient	Description	Subprocess	Act	Timestamp
e_1	101	Visit	Contact	C_Vi	10-10-2019
e_2	101	Calcium	Labtest	L_Ca	11-10-2019
e_3	101	Register	Contact	C_Re	12-10-2019
e_4	101	Glucose	Labtest	L_Gl	13-10-2019
e_5	101	Consultation	Contact	C_Cs	14-10-2019
e_6	101	Consultation	Contact	C_Cs	15-10-2019
e_7	102	Register	Contact	C_Re	16-10-2019
e_8	102	Glucose	Labtest	L_Gl	17-10-2019
...	...	...	...	...	...

results are presented in Sect. IV. Sect. V discusses related work, and Sect. VI concludes the paper .

II. Activity Tree and Process Hierarchy

Preliminaries: Let $\sigma = \langle a_1, \cdots, a_n \rangle \in \Sigma^*$ be a sequence of activities, which is also called a *trace*. Let a multiset $L \subseteq \mathbf{B}(\Sigma^*)$ of traces be an *event log*. Tab. I shows a list of events; Fig. 2(a) shows the sequential traces, in a graphical representation, where each square represents an event that is labeled with an activity. Given an event log, a process discovery algorithm D automatically constructs a process M (e.g., in Petri net notation). The quality of such a model M can assessed with respect to the log L using four quality dimensions: fitness, precision, generalization, and complexity [8].

To discover a hierarchical process model, we leverage a rather known concept that represents the process hierarchical information and call it *activity tree*. We define an activity tree as the hierarchical relations between activities of a process. Formally, an activity tree is a non-overlapping, hierarchical clustering of activities.

Definition 1 (Activity tree)**.** Let Σ be a set of activities and L an *event log* over Σ. Let A be a superset of Σ, i.e., $\Sigma \subset A$. Function $\gamma : A \to \mathcal{P}(A)$ is a mapping that maps each activity $x \in A$ to a set of activities $X \subset A$ as the children of x. An activity tree (A, γ) is valid for L, if and only if:

1) The children of any two labels do not overlap, i.e., for each $x, y \in A$, $x \neq y \Leftrightarrow \gamma(x) \cap \gamma(y) = \emptyset$.
2) The union of the leaves is Σ, i.e., the set of activities that occurred in log L, i.e., $\{x \in A \mid \gamma(x) = \emptyset\} = \Sigma$.
3) The tree (A, γ) is connected, i.e., for each $x \in A$, either there is $y \in A$ such that $x \in \gamma(y)$, or x is the *root*.

Note that constraints (2) and (3) imply that for all $x \in A$, $x \notin \gamma(x)$. Furthermore, it is also possible to define an activity tree as a directed acyclic graph where each node represents an activity and has only one incoming edge, except the *root* node, which does not have any incoming edge.

Example 1. Fig. 2(b) exemplifies the activity tree (A, γ), where $A = \{\texttt{root}, \texttt{C}, \texttt{L}, \texttt{S}, \texttt{Vi}, \texttt{Cs}, \texttt{Re}, \texttt{Ca}, \texttt{Gl}, \texttt{Cr}, \texttt{Or}, \texttt{Pr}, \texttt{Op}\}$, and, for example, $\gamma(\texttt{C}) = \{\texttt{Vi}, \texttt{Cs}, \texttt{Re}\}$.

We call the node that has no parent in γ the root node, and the corresponding process the *root process*. For each $a \in A$, $\gamma(a) \neq \emptyset$, we call it a (sub)process a with $\gamma(a)$ as its activities.

We define the height of a node in the activity tree, which is later used to recursively compute the abstracted logs bottom-up. Let $height : A \to \mathbb{N}^0$ be a function that maps each $x \in A$ to the height (a non-negative integer) of x in the activity tree (A, γ). For each $x \in A$, if $\gamma(x) = \emptyset$, then $height(x) = 0$, else $height(x) = 1 + Max_{c \in \gamma(x)} height(c)$. A process model with a maximal height of an activity tree is 1, is called *flat*.

III. The FlexHMiner Algorithm

In this section, we discuss the three steps of the FlexHMiner approach: (1) compute an activity tree, (2) compute abstracted logs, and (3) compute subprocess models.

A. Computing Activity Tree

For the first step, computing an activity tree, we present three different methods: one supervised using domain knowledge, one automated using random clustering, and one fall-back using flat tree.

1) Using Domain Knowledge: Domain knowledge can be used, as a gold standard, to create an activity tree. This can be done manually. In other situations, the log itself may already contain an encoding of human knowledge that can be utilized in creating the activity tree. In Tab. I, the column *Act* demonstrates such an encoding of hierarchy. For instance, the activity label `C_Vi`, indicating that the event belongs to subprocess `C`. Following the same strategy as the State Chart

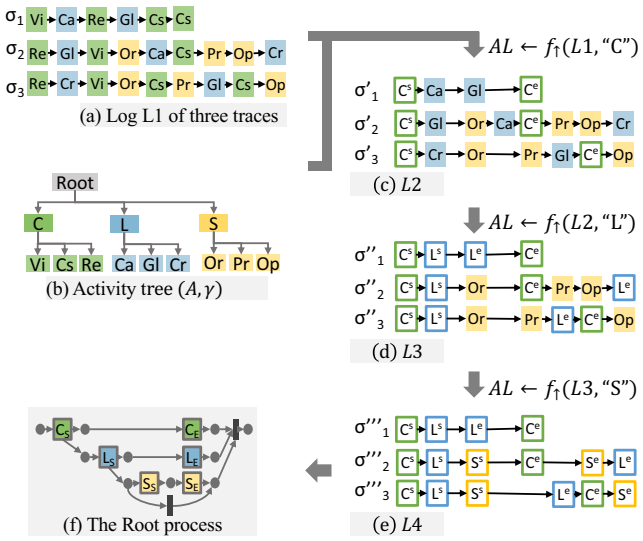


Fig. 2: FlexHMiner applied on the running example.

Miner (SCM) [14], a simple text parser is built to split the labels of activities and to convert the domain knowledge into an activity tree. For our experiment, we found seven real-life logs of two processes, where such information regarding the hierarchy is readily encoded in the activity labels [2] [3].

2) Using Random Clustering: Let $L \subset \mathbf{B}(\Sigma^*)$ be an event log. Let $maxSize \geq 2$ be the indicated maximal size of subprocesses. The random clustering algorithm, as listed in Algorithm 1, takes Σ and $maxSize$ as inputs and creates random subprocesses of size less than $maxSize$. It first initializes the activity tree (A, γ) using Σ (see Line 1). It then uses $maxSize$ to determine the number n of subprocesses at the current height (see Line 2-5). Next, it creates n parent nodes as P and simply assigns each activity $c \in C$ randomly to a cluster $p \in P$ (see Line 6-12), while updating A and γ accordingly (see Line 8 and 11). To decide whether to continue with the next height, the current set of parent nodes P becomes the set of child nodes C (see Line 13): if the size of C (i.e., $\mid C \mid$) is greater than $maxSize$, then the algorithm creates another level of parent nodes (as abstracted processes); otherwise, the *while* loop terminates, and the root node is added (see Line 15 and 16). In this paper, $maxSize$ is set to 10 as default. We use the random clustering to show an unsupervised way to create an *activity tree* and to investigate a possible baseline for the qualities of hierarchical models.

3) Using Flat Tree: Given a log L over activities Σ, a flat activity tree (A, γ) (with height 1) is constructed as follows. We have $A = \Sigma \cup \{root\}$. For all $a \in \Sigma$, $\gamma(a) = \emptyset$; and for $root \in A$, $\gamma(root) = \Sigma$. We use the flat tree approach as a fall-back.

[2]See the description of the BPI Challenge 2012 at https://www.win.tue.nl/bpi/doku.php?id=2012:challenge: *"The event log is a merger of three intertwined sub processes. The first letter of each task name identifies from which sub process (source) it originated."*

[3]See the description of the BPI Challenge 2015 at https://doi.org/10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1.: *"The first two digits as well as the characters* [of the activity label] *indicate the subprocess the activity belongs to. For instance ... `01_BB_xxx` indicates the 'objections and complaints' ('Beroep en Bezwaar' in Dutch) subprocess."*

B. Computing Abstracted Logs and Models

Given an activity tree (A, γ) and a log L, the second step uses the projection ($f_{\downarrow}$) and abstraction ($f_{\uparrow}$) functions to recursively compute sublogs for each non-leaf node in the tree. We first discuss the projection and abstraction functions, after which we present the algorithm.

1) Projection: Given a log L, an activity tree (A, γ), and a subprocess $sp \in A$, the projection of L on the activities $\gamma(sp)$ of sp is rather straightforward and standard, which allows us to create a corresponding log for sp. The projection function simply retains the events of activities $\gamma(sp)$ and remove the rest. Formally, given a trace $\sigma = \langle e \rangle \cdot \sigma'$: if $e \in \gamma(sp)$, $f_{\downarrow}(\sigma, sp) = \langle e \rangle \cdot f_{\downarrow}(\sigma', sp)$, otherwise, $f_{\downarrow}(\sigma, sp) = \langle\rangle \cdot f_{\downarrow}(\sigma', sp)$. We overload the function for any log L: $f_{\downarrow}(L, sp) = \bigcup_{\sigma \in L} f_{\downarrow}(\sigma, sp)$, if $f_{\downarrow}(\sigma, sp) \neq \langle\rangle$.

Example 2. For example, given the (simplified) trace $\sigma_1 = \langle \texttt{Vi}, \texttt{Ca}, \texttt{Re}, \texttt{Gl}, \texttt{Cs}, \texttt{Cs} \rangle$, the subprocess `C`, and the activity-hierarchy γ where $\gamma(\texttt{C}) = \{\texttt{Vi}, \texttt{Cs}, \texttt{Re}\}$ (as shown in Fig. 2(a) and (b)) then the trace of `C` is computed as $f_{\downarrow}(\sigma_1, \texttt{C}) = \langle \texttt{Vi}, \texttt{Re}, \texttt{Cs}, \texttt{Cs} \rangle$.

2) Abstraction: The *abstraction function* $f_{\uparrow}$ returns the abstracted, intermediate log after abstracting (removing) the internal behavior of a subprocess sp. Essentially, the abstraction function hides irrelevant internal behavior by not retaining the detailed events of a subprocess; it only keeps the *relevant behavior*. In this paper, we consider both the *start* and the *end events* of a subprocess as the relevant behavior. In this way, the abstracted log only records when a subprocess starts and ends[4]. If a subprocess `x` has different start activities (e.g., `Vi` and `Re`), they are abstracted into a single start activity $\texttt{x}^s$

[4]Note that other abstraction functions can be conceived, e.g., a function that will only render the start or the end event of the subprocess.

Algorithm 1 Compute random tree (A, γ)

Input: the input log L over Σ, and the maximal size of a process $maxSize$
Output: activity tree (A, γ),
1: $A \leftarrow \Sigma$; and **for** $a \in \Sigma$ **do** $\gamma(a) \leftarrow \emptyset$ {*initiate A and the leaves of γ*}
2: $C \leftarrow \Sigma$ {*Use C to represent the child activities at the current height*}
3: **while** $\mid C \mid > maxSize$ **do**
4: {*create parent clusters P to ensure the size of a process is at most $maxSize$, and apply clustering to assign each $c \in C$ to a parent cluster p*}
5: $n \leftarrow \lfloor (\mid C \mid -1)/maxSize \rfloor + 1$ {*Calculate the number of parents*}
6: Create labels $p_1, \cdots, p_n$ {*Create n random parent nodes/processes*}
7: Initiate $P \leftarrow \{p_1, \cdots, p_n\}$: **for** $p \in P$ **do** $\gamma(p) \leftarrow \emptyset$
8: $A \leftarrow A \cup P$
9: **for** $c \in C$ **do**
10: Select a random $p \in P$, where $\mid \gamma(p) \mid < maxSize$
11: $\gamma(p) \leftarrow \gamma(p) \cup \{c\}$
12: **end for** {*all children have been assigned to a parent process*}
13: $C \leftarrow P$
14: **end while** {$\mid C \mid \leq maxSize$}
15: $A \leftarrow A \cup \{root\}$ {*Create the root node/process and add to A*}
16: **for** $c \in C$ **do** $\gamma(root) \leftarrow \gamma(root) \cup \{c\}$ {*Assign $c \in C$ to the root process*}
17: **return** (A, γ)

(e.g., $\mathtt{C}^s$, see Fig. 2(c)). The same holds for the end activities, which are abstracted into a single end activity $\mathtt{x}^e$.

Let $SP = \gamma(sp)$ denote the set of activities of subprocess sp. Let I_s denote the index of the first event of sp in σ as $I_s = \min_{e \in \sigma \wedge e \in SP} \sigma[e]$. Similarly, we use I_e to refer to the index of the last event of sp in σ, i.e., $I_e = \max_{e \in \sigma \wedge e \in SP} \sigma[e]$. We define $f_\uparrow$ as follows:

$$f_\uparrow(\sigma, SP) = \begin{cases} \langle e \rangle \cdot f_\uparrow(\sigma', SP) & e \notin SP \\ \langle sp^s \rangle \cdot f_\uparrow(\sigma', SP) & e \in SP, \sigma[e] = I_s \\ \langle sp^e \rangle \cdot f_\uparrow(\sigma', SP) & e \in SP, \sigma[e] = I_e \\ \langle \rangle \cdot f_\uparrow(\sigma', SP) & e \in SP \\ & \wedge \quad I_s < \sigma[e] < I_e \end{cases} \quad (1)$$

We overload the function for any log L, i.e., $f_\uparrow(L, sp) = \bigcup_{\sigma \in L} f_\uparrow(\sigma, sp)$ [5].

Example 3. For example, after abstracting subprocess C from σ_1, we have $f_\uparrow(\sigma_1, \mathtt{C}) = \langle \mathtt{C}^s, \mathtt{Ca}, \mathtt{Gl}, \mathtt{C}^e \rangle$. Similar, the trace after abstracting subprocess L is $f_\uparrow(\sigma_1, \mathtt{L}) = \langle \mathtt{Vi}, \mathtt{L}^s, \mathtt{Re}, \mathtt{L}^e, \mathtt{Cs}, \mathtt{Cs} \rangle$. To obtain the parent-trace, we apply $f_\uparrow$ recursively; for instance, $f_\uparrow(f_\uparrow(\sigma_1, \mathtt{C}), \mathtt{L}) = f_\uparrow(f_\uparrow(\sigma_1, \mathtt{L}), \mathtt{C}) = \langle \mathtt{C}^s, \mathtt{L}^s, \mathtt{L}^e, \mathtt{C}^e \rangle = \sigma_1''$, see Fig. 2(d).

The abstraction function $f_\uparrow$ is commutative and associative for *non-related* subprocesses[6]. These two properties allow Algorithm 2 to use the abstraction function in a recursive, bottom-up manner and to go iteratively through the nodes. The algorithm creates an abstracted log at each height of the activity tree, which is discussed in the next section.

3) Recursion: The algorithm computes the log mapping α and the model mapping β with three inputs: (1) an event log L, (2) an activity tree (A, γ), and (3) a flat process discovery algorithm D. It applies the projection function bottom-up for each subprocess on the abstracted log AL, starting with height 1 until but does not include the *root* process (see Lines 2 - 13). For every subprocess $p \in P$ of the same height, the algorithm applies the projection function to obtain the sublog L_p for p (Line 8) and updates the log mapping and model mapping (see Lines 9 and 10). The algorithm then computes the intermediate, abstracted log AL by using the abstraction function $f_\uparrow$ (see Line 11). The abstracted log AL is updated after each $p \in P$ (see Lines 5 - 12) at height i. The algorithm terminates once it has completed processing the root process. It returns a *hierarchical model* $(A, \gamma, \alpha, \beta)$, where the log mapping α maps each (sub)process $p \in A$, $\gamma(p) \neq \emptyset$, to the associated log L_p, and the model mapping β maps each p to the associated model $M_p = D(L_p)$ of (sub)process p.

Example 4. Given the log $L1$ and activity tree (A, γ)), respectively, shown in Fig. 2(a) and (b), the three subprocesses C, L, and S have height 1. Fig. 2(c), (d), and (e) show the abstracted log AL obtained after each iteration of the inner-loop at Lines 5-11. For example, in the first iteration (see Fig. 2 (c)), applying $f_\uparrow(L1, \mathtt{C})$ on log $L1$, we obtain $AL = L2$ where the activities of C is removed from $L2$ and only the start and the end of C are retained (as discussed in Example 3). In the second iteration, the algorithm continues with log AL and applies $f_\uparrow(AL, \mathtt{L})$, see Fig. 2 (d).

[5]For the sake of brevity, we consider the labels $a = a^s = a^e$ for both projection and abstraction functions and only distinguish them when we apply a discovery algorithm.

[6]Two nodes are non-related if they do not share ancestors or descendants in the activity tree)

TABLE II: Statistical information of the event logs.

Data	#acts	#evts	#case	#dpi	avg e/c	max e/c
BPIC12 *	36	262,200	13,087	4,366	20	175
BPIC17f *	18	337,995	21,861	1,024	18	32
BPIC15_1f *	70	21,656	902	295	24	50
BPIC15_2f *	82	24,678	681	420	36	63
BPIC15_3f *	62	43,786	1,369	826	32	54
BPIC15_4f *	65	29,403	860	451	34	54
BPIC15_5f *	74	30,030	975	446	31	61

IV. Empirical Evaluation

We implemented the FlexHMiner approach in the process mining toolkit ProM[7]. We evaluated the quality of the models created by FlexHMiner using seven real-life data sets and compared the results.

In the following, we discuss the data sets and the experimental setup, followed by our empirical analysis. All experiments are run on an Intel Core i7- 8550U 1.80GHZ with a processing unit of a 16 GB HP-Elitebook running Windows 10 Enterprise.

A. Data sets

An overview of the statistical information, including the number of distinct activities (acts), events (evts), cases, distinct sequences (dpi), and of events per case (e/c) of the seven logs, is given in Tab. II. These logs are the filtered logs used in [8] as a benchmark[8], which allows us to compare our result with the qualities of the flat models reported in [8].

[7]http://www.promtools.org/. The source code and results: github.com/xxlu/prom-FlexHMiner

[8]https://doi.org/10.4121/uuid:adc42403-9a38-48dc-9f0a-a0a49bfb6371

Algorithm 2 Compute log hierarchy α and models β

Input: Log L, activity tree (A, γ), and discovery algorithm D
Output: Log mapping α and model mapping β
1: $maxHeight \leftarrow Max_{c \in A} height_\gamma(c)$ {*calculate the maximal height of the tree*}
2: $AL \leftarrow L$
3: **for** $i = 1$ to $maxHeight - 1$ **do**
4: $P \leftarrow \{p \in A \mid height(p) = i\}$
5: **for** $p \in P$ **do**
6: {*Iteratively go through each $p \in P$ at the same height i and perform log projection and abstraction*}
7: $C \leftarrow \gamma(p)$ {*get the activities C of subprocess p*}
8: $L_p \leftarrow f_\downarrow(AL, C)$ {*project the log so far on the activities C*}
9: $\alpha(p) \leftarrow L_p$
10: $\beta(p) \leftarrow D(L_p)$
11: $AL \leftarrow f_\uparrow(AL, p)$ {*abstract the log so far using the activities C }*
12: **end for** {*the log so far AL has been abstracted from P to height i }*
13: **end for** {$i == maxHeight$}
14: $\alpha(root) \leftarrow AL$ {*map the root to the resulted abstracted log*}
15: $\beta(root) \leftarrow D(AL)$ {*map the root to the discoverede model*}
16: **return** $(A, \gamma, \alpha, \beta)$

B. Experimental Setups

For computing activity tree, we used the three methods: random (RC-FH), flat tree (F-FH), and domain knowledge (DK-FH). For computing hierarchical models, we use two state-of-the-art process discovery algorithms as D, namely the Inductive Miner with 0.2 path filtering (IMf) and the Split Miner (SM) [7] with standard parameter settings. According to the recent work of Augusto et al. [8], these two algorithms outperform others in terms of the four quality measures and execution time.

We run these six configurations on the seven data sets. For each log and for each of the two flat discovery algorithms, we also report the results in [8], for which we use F* to represent. Thus, in total eight rows for each data set.

C. Model Quality Measures

We assess the quality of a model M, with respect to a log L, using the following four dimensions:

- For *fitness* $fi(L, M) \in [0, 1]$, we use the alignment based fitness defined by Adriansyah et al. [15], also used in [8].
- For *precision* ($pr(L, M) \in [0, 1]$), we use the measure defined in [16], known as ETC-align.
- We also compute the F1-score, which is the harmonic mean of fitness and precision: $f1(M, L) = 2 * \frac{fi(L,M)*pr(L,M)}{fi(L,M)+pr(L,M)}$.
- For *generalization*, we follow again the same approach adopted in [8]. We divide the log into $k = 3$ subsets: L is randomly divided into L_1, L_2, L_3, $ge(L, M) = \frac{1}{3}\sum_{1\leq i\leq 3} 2 * \frac{fi(L_i,M_i)*pr(L,M_i)}{fi(L_i,M_i)+pr(L,M_i)}$ where M_i corresponds to L_i.
- Finally, for *complexity*, we report the *size* (number of nodes) and the Control-Flow Complexity (CFC) of a model [8]. Let $PN = (P, T, F, l, m_i, m_f)$ be a Petri Net. $Size(PN) =\mid P \mid + \mid T \mid$.

$$CFC(PN) = \sum_{t\in T\wedge(|^\bullet t|>1\vee|t^\bullet|>1)} 1 + \sum_{p\in P\wedge(|^\bullet p|>1\vee|p^\bullet|>1)} \mid p^\bullet \mid$$

Let $(A, \gamma, \alpha, \beta)$ be a hierarchical model returned. Let $q \in \{fi, pr, F1, Ge, CFC, Size\}$ be any quality measure that takes a model M and/or a log L as inputs and returns the quality value of the model. We calculate and report the average quality measure $\overline{q}$ as follows : $\overline{q}(A, \gamma, \alpha, \beta) = avg_{a\in A\wedge\gamma(a)\neq\emptyset} q(\beta(a), \alpha(a))$. For example, $\overline{fi}(A, \gamma, \alpha, \beta)$ is calculated as the average value of the individual fitness values of each subprocess in the hierarchical model.

The computation of fitness, precision, and generalization uses a state-of-the-art technique known as alignment [15][9]. As mentioned, we also list the quality values reported by Augusto et al. using "F*" [8]. Since we were unable to compute all qualities of the flat models, and for the sake of consistency, we mainly discuss our results with respect to the results of "F*".

[9]A 60 minute time-out limit is set for computing the alignment of each model. When the particular quality measurement could not be obtained due to either syntactical or behavioral issues in the discovered model or a timeout when computing the quality measures, we record it using the "−" symbol.

TABLE III: Evaluation results of models for the seven logs.

Data	CAlg	DAlg	$\overline{Fi}$	$\overline{Pr}$	$\overline{F1}$	$\overline{Ge}$	$\overline{CFC}$	$\overline{Size}$	#SPs	$\overline{\#Act}$
BPIC12	F-FH		-	-	-	-	66	115	1	24
	F*		**0.98**	0.50	0.66	0.66	37	59	1	-
	DK-FH	IMf	0.96	**0.78**	**0.86**	0.85	**20**	36	4	8
	RC-FH		0.97	**0.78**	**0.86**	0.88	20	**35**	4	8
	F-FH		**0.97**	0.55	0.70	0.69	53	89	1	24
	F*		0.75	0.76	0.75	0.76	32	53	1	-
	DK-FH	SM	0.89	**0.94**	**0.91**	**0.91**	**10**	**22**	4	8
	RC-FH		0.92	0.90	0.90	0.90	14	28	3	8
BPIC17f	F-FH		**0.98**	0.57	0.72	0.73	20	51	1	18
	F*		**0.98**	0.70	0.82	0.82	20	35	1	-
	DK-FH	IMf	0.97	**0.98**	**0.97**	**0.97**	**6**	**17**	4	6
	RC-FH		**0.98**	0.90	0.94	0.92	9	22	3	7
	F-FH		**0.98**	0.63	0.76	0.76	23	47	1	18
	F*		0.95	0.85	0.90	0.90	17	32	1	-
	DK-FH	SM	0.96	**0.98**	**0.97**	**0.97**	**6**	**18**	4	6
	RC-FH		**0.98**	0.92	0.95	0.95	8	20	3	7
BPIC15_1f	F-FH		0.96	0.36	0.52	0.51	128	217	1	70
	F*		0.97	0.57	0.72	0.72	108	164	1	-
	DK-FH	IMf	**0.99**	**0.87**	**0.91**	**0.93**	**9**	**19**	15	6
	RC-FH		0.97	0.84	0.88	0.88	16	29	9	10
	F-FH		0.93	0.87	0.90	0.90	64	152	1	70
	F*		0.90	0.88	0.89	0.89	43	110	1	-
	DK-FH	SM	**0.96**	0.98	**0.97**	**0.97**	**4**	**14**	15	6
	RC-FH		0.88	**0.99**	0.93	0.93	8	22	9	10
BPIC15_2f	F-FH		-	-	-	-	183	313	1	82
	F*		0.93	0.56	0.70	0.70	123	193	1	-
	DK-FH	IMf	**0.97**	**0.91**	**0.93**	**0.93**	**7**	**16**	21	5
	RC-FH		0.95	0.85	0.89	0.88	16	33	10	10
	F-FH		0.83	0.88	0.85	0.85	72	198	1	82
	F*		0.77	0.90	0.83	0.82	41	122	1	-
	DK-FH	SM	**0.94**	**0.99**	**0.97**	**0.97**	**4**	**13**	21	5
	RC-FH		0.83	0.96	0.89	0.89	11	26	10	10
BPIC15_3f	F-FH		-	-	-	-	136	244	1	62
	F*		0.95	0.55	0.70	0.69	108	159	1	-
	DK-FH	IMf	**0.97**	**0.94**	**0.95**	**0.95**	**6**	**15**	17	5
	RC-FH		0.94	0.87	0.90	0.89	16	33	8	10
	F-FH		0.81	0.92	0.86	0.87	50	135	1	62
	F*		0.78	0.94	0.85	0.85	29	90	1	-
	DK-FH	SM	**0.95**	**0.99**	**0.97**	**0.97**	**3**	**12**	17	5
	RC-FH		0.87	0.98	0.92	0.92	10	23	7	9
BPIC15_4f	F-FH		-	-	-	-	121	242	1	65
	F*		0.96	0.58	0.72	0.72	111	162	1	-
	DK-FH	IMf	**0.98**	**0.96**	**0.97**	**0.97**	**5**	**13**	21	4
	RC-FH		0.97	0.81	0.87	0.87	18	35	8	10
	F-FH		0.79	0.92	0.85	0.85	51	147	1	65
	F*		0.73	0.91	0.81	0.80	31	96	1	-
	DK-FH	SM	**0.95**	**0.99**	**0.97**	**0.97**	**2**	**11**	21	4
	RC-FH		0.83	0.98	0.90	0.89	8	24	8	10
BPIC15_5f	F-FH		-	-	-	-	142	255	1	74
	F*		0.94	0.18	0.30	0.61	95	134	1	-
	DK-FH	IMf	**0.98**	**0.95**	**0.97**	**0.96**	**5**	**14**	21	5
	RC-FH		0.98	0.80	0.87	0.87	19	34	9	10
	F-FH		0.85	0.91	0.88	0.88	54	163	1	74
	F*		0.79	0.94	0.86	0.85	30	102	1	-
	DK-FH	SM	**0.96**	**1.00**	**0.98**	**0.98**	**2**	**11**	20	5
	RC-FH		0.83	0.98	0.90	0.90	8	23	9	10

On the seven starred data sets, there are two subprocesses for RC-FH-SM, one for DK-FH-SM, and none for DK-FH-IMf and RC-FH-IMf, where the returned alignments were indicated unreliable. As a result, the fitness of these models are -1, and subsequently, the precision and generalization cannot be computed. The results of these models are excluded when computing the average quality scores.

D. Results and Discussion

Tab. III summarises the results of our evaluation. For each data set (*Data*) and each of the two discovery algorithms (*DAlg*), we obtained four (hierarchical) models and report on the results. To also provide concrete examples, Fig. 3(a) shows the root process and three subprocesses obtained by

applying DK-FH-IMf on the BPIC15_1f log, while the other subprocesses are hidden (abstracted). By contrast, Fig. 3(b) shows the flat model by F-FH-IMf on the same log.

Fitness. Column $\overline{Fi}$ (Tab. III) reports on the average fitness of the models discovered. For both IMf and SM, in 5 of the seven logs, the hierarchical models returned by DK-FH achieved a better fitness score than F*. More specifically, there is on average an increase of 0.015 (2%) in fitness, when compared the fitness of F* to DK-FH. Considering SM, the improvement in fitness is more significant: the average increase in fitness is 17%, from 0.810 to 0.946. The moderate improvement in fitness for IMf is due to the fact that IMf tends to optimize fitness. As the average fitness of the seven models is already very high (0.959), there is little room for improvement.

Precision. Column $\overline{Pr}$ (Tab. III) lists the average precision of the models. When comparing the average precision scores of the subprocesses (DK-FH) to the flat models (F*), DK-FH has achieved a considerable improvement in precision, especially for IMf. For all seven logs, the subprocesses obtained by both DK-FH-IMf and DK-FH-SM have achieved an average precision higher than the F* approaches: for DK-FH-IMf, the average precision is increased by 75.4% (from 0.520 to 0.912); for DK-FH-SM, there is also an 11.2% increase in the precision scores (from 0.883 to 0.982).

An explanation for such significant improvements in the average precision can be the following. As the subprocesses are relatively small and the sublogs do not contain interleaved behavior of other concurrent subprocesses, it allowed the discovery algorithm to discover rather sequential models of high precision, while maintaining the high fitness (see Fig. 3 (b), (c) and (e)). When a subprocess itself contains much concurrent behavior (for example, see Fig. 3 (d)), the highly concurrent, flexible behavior is then localized within this one subprocess, without affecting the models of other subprocesses.

F1 score. For both IMf and SM and for all seven logs, the average F1 scores of the subprocesses of DK-FH outperform the ones of F*. This is mainly due to the improvement in both fitness (for SM) and precision (for IMf), which led to the increase in the harmonic mean of fitness and precision. On average DK-FH achieved an increase of 41.9% (from 0.660 to 0.936) for IMf and 14.4% for SM (from 0.842 to 0.962) in the F1 scores over the seven logs.

Fig. 5 shows the distribution of the F1-scores of the models of each approach in more detail (IMf on the left and SM on the right). Compared to the F1 scores of the models by F-FH (see purple lines), DK-FH (blue boxplot and dots) always scores higher. Compared to the F* results, there are only three exceptional cases for IMf. We looked into these models of IMf and found that many activities are put in parallel. Thus, fitness was very high (e.g., 0.972) but precision was very low (e.g., 0.335). Interestingly, for the exact same subprocess, SM was able to discover a much more sequential model. This model still has a high fitness (0.958) but a much higher precision (1.00) and also a high generalization (0.979). This can be an interesting future work: since DK-FH allows for the use of any discovery algorithm, one can design an algorithm that selects the best algorithm (model) for each subprocess to further increase the quality of the hierarchical models.

Generalization. Column $\overline{Ge}$ (Tab. III) reports on the generalization scores (using 3-fold cross validation). In all seven logs, for both IMf and SM, there is overall an increase of 23.2% (from 0.771 to 0.950) in the generalization scores (33.3% for IMf and 14.8% for SM), when comparing DK-FH to F*.

Complexity. Since the subprocesses are much smaller than the flat model, the average complexity scores of subprocesses of DK-FH are, as expected, significantly lower than F-FH or F*. For DK-FH-IMf, the CFC is decreased by 90.6% (from 86.0 to 8.1), and 86.3% for DK-FH-SM (from 31.9 to 4.4), when compared to the CFC scores of the models of F* on the seven logs. The improvement is even more significant when it is compared to the models by F-FH.

Fig. 6 shows the distribution of CFC of the submodels by two approaches in more detail: DK-FH (blue boxplots) and F-FH (purple lines). It shows the significant decrease in CFC when using DK-FH.

Discussion. Overall, the results on the real-life logs have shown that the quality of the submodels returned by the DK-FH approach are higher than the one returned by either the random clustering approachs or the flat discovery algorithms. For example, compared to F*, DK-FH achieved an increase of 0.07 in fitness, 0.25 in precision, and 0.18 in generalization, on average. As expected, the DK-FH outperforms the RC-FH.

Interestingly, the random activity clustering approach (RC-FH) also shows relatively consistent improvements in the qualities of obtained submodels, compared to the flat tree approach and the flat discovery (F*), as listed in Tab. III and shown in Fig. 5. This may suggest that discovering hierarchical models may have a certain beneficial factor in terms of model qualities, comparing to discovering a complex, flat model. Additional experiments are needed to validate this observation. However, if this is indeed the case, this may allow future process discovery algorithms to focus on small processes (say less than 10-20 activities), while designing other algorithms for clustering the activities and discovering the process hierarchy.

A related, interesting observation is that for some subprocesses and their sublogs, SM was able to discover better models than IMf, and in other cases, IMf was able to find better models than SM. Since FlexHMiner allows for the use of any discovery algorithm, this suggests that one can build in a selection strategy of the best subprocesses to further increase the quality of the hierarchical models, as future work.

We also observed that it is rather trivial to flatten the full model or certain subprocesses of interest, while allowing abstracting the others, see Fig. 4. For example, in Fig. 2(c) and (d), the logs $L2$ and $L3$ respectively show abstracting one or two subprocesses. Applying a discovery algorithm on $L2$ returns a model where subprocess `C` is abstracted; and for $L3$, a model is returned where subprocesses `C` and `L` are abstracted, while the detailed activities of subprocess `S` are retained.

Fig. 3: The flat model discovered using F-FH-IMf on the BPIC15_1f log.

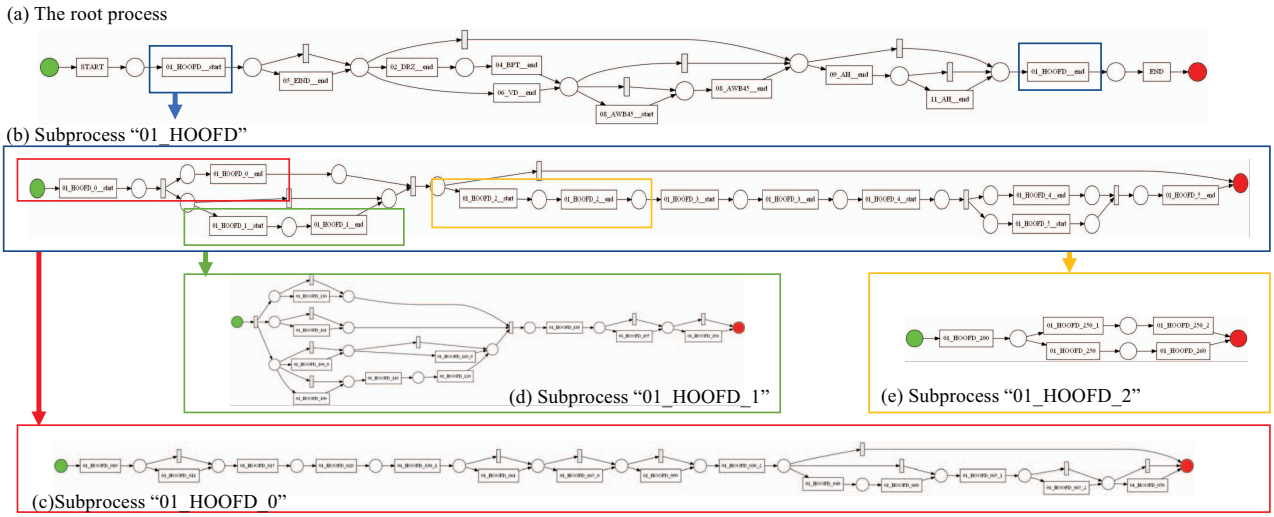


Fig. 4: The root process and three subprocesses discovered by DK-FH-IMf on the BPIC15_1f log.

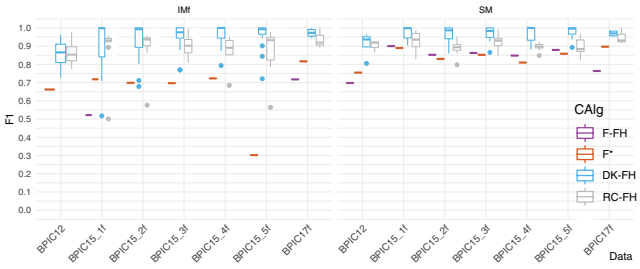


Fig. 5: Significantly higher F1-scores achieved by DK-FH (blue), followed by RC-FH (grey), compared to the F-FH (purple) and the results reported in [8] (purple), on the seven benchmark data sets (on the left IMf, and on the right SM).

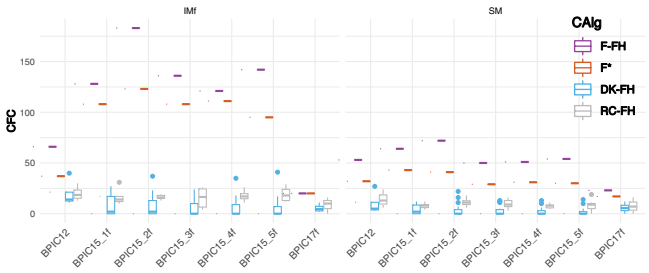


Fig. 6: A significant decrease in the complexity (CFC) achieved by the DK-FH (blue), compared to the F-FH (F) returned (purple lines) for the 16 data sets.

V. Related Work

In this section, we discuss related works regarding hierarchical process discovery and event abstraction, specifically.

Following the unsupervised strategy, Bose and van der Aalst [9] are one of the first who propose to detect reoccurring consecutive activity sequences as subprocesses. This approach treats concurrent subprocesses as running sequentially and thus does not assume true concurrent/interleaving subsprocesses. Later, Tax et al. [17] propose an approach to generate frequent subprocesses, also known as *local process models*. This approach enables detecting interleaving subprocess in an unsupervised manner, which is used to create abstracted logs.

Using the supervised strategy, the State Chart Miner (SCM) [14] extends Inductive Miner (IM) and uses information in the activity labels to create a hierarchy to discover hierarchical models. SCM also focuses on sequential subprocesses, where non-consecutive events of the same instance are cut into separate traces. This assumption leads to the discovery of models that are overly segmented and fail to capture concurrent behavior, as shown in Fig. 1a. Furthermore, SCM can only use IM for discovering subprocesses.

Mannhardt et al. [13] require users to define complete behavioral models of subprocesses and their relations to compute abstracted logs. This prerequisite of specifying the full behavior of low-level processes put much burden on the users. Moreover, it uses the alignment technique to compute high-level logs which is computationally very expensive and can be nondeterministic.

Other approaches assume additional attributes to indicate the hierarchical information. Using a relational database as

input, Conforti et al. [12] assume that each event has a set of primary and foreign keys that can be used as the subprocess instance identifier, in order to determine subprocesses and multi-instances. However, such event attributes may not be common in most of the event logs. As an alternative, Wang et al. [11] assume that the events contain start and complete timestamps and have explicit information of the follow-up events (i.e., the next intended activity, which they called "transfer" attributes). Senderovich et al. [18] propose the use of patient schedules in a hospital as an approximation to the actual life-cycle of a visit. These approaches cannot be applied in our case, since we do not find these attributes (such as explicit causal-relations between events, the start and complete timestamps, or the scheduling of activities) in our logs.

A very recent literature review conducted by van Zelst et al. [19] provides an extensive overview and taxonomy for classifying different event abstraction approaches. In [19], Zelst et al. studied 21 articles in depth and compared them among seven different dimensions. One dimension is particularly important to distinguish our approaches, namely the supervision strategy. None of the 21 methods enable both supervised and unsupervised, whereas we have shown that we can used both supervised (domain knowledge) and unsupervised (random clustering) to discover an activity tree for event abstraction. Our evaluation has shown FlexHMiner to be flexible and applicable in practice. It is worthy to mention that our approach does assume that for each case, each subprocess is executed at most once (i.e., single-instance), while a subprocess can contain loops. As future work, we can extend the algorithm to include, in addition to the abstraction and projection functions, the multi-instance detection and segmentation techniques to handle multi-instance subprocesses.

VI. Conclusion

In this work, we investigated the hierarchical process discovery problem. We presented FlexHMiner, a general approach that supports flexible ways to compute process hierarchy using the notion of *activity tree*. We demonstrate this flexibility by proposing three methods for computing the hierarchy, which vary from fully supervised, using domain knowledge, to fully automated, using a random approach. We investigated the quality of hierarchical models discovered using these different methods. The empirical evidence, using seven real-life logs, demonstrates that the one using supervised approach outperforms the one using random clustering in terms of the four quality dimensions. But both methods outperform the flat model approaches, which clearly demonstrates the strengths of the FlexHMiner approach.

For future work, we plan to investigate different algorithms for computing activity trees to further improve the quality of hierarchical models. Also, the concept of activity trees can be extended to data-aware or context-aware activity trees. For example, an activity node can be associate with a data contraint (context), so that the events labeled with the same activity but have different data attributes (contexts) can be abstracted into different subprocesses.

Acknowledgment

This research was supported by the NWO TACTICS project (628.011.004) and Lunet Zorg in the Netherlands. We would also like to thank the experts from the VUMC for their extremely valuable assistance and feedback in the evaluation.

References

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

[2] X. Lu, S. A. Tabatabaei, M. Hoogendoorn, and H. A. Reijers, "Trace clustering on very large event data in healthcare using frequent sequence patterns," in *BPM*, ser. Lecture Notes in Computer Science, vol. 11675. Springer, 2019, pp. 198–215.

[3] H. A. Reijers, J. Mendling, and R. M. Dijkman, "Human and automatic modularizations of process models to enhance their comprehension," *Inf. Syst.*, vol. 36, no. 5, pp. 881–897, 2011.

[4] 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, Beijing, China, August 26, 2013*, 2013, pp. 66–78.

[5] J. Carmona and J. Cortadella, "Process discovery algorithms using numerical abstract domains," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 12, pp. 3064–3076, 2014.

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

[7] 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 Computer Society, 2017, pp. 1–10.

[8] 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," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 4, pp. 686–705, 2019.

[9] R. J. C. Bose and W. M. P. van der Aalst, "Abstractions in process mining: A taxonomy of patterns," in *BPM*, ser. LNCS, vol. 5701. Springer, 2009, pp. 159–175.

[10] F. M. Maggi, T. Slaats, and H. A. Reijers, "The automated discovery of hybrid processes," in *BPM*, ser. Lecture Notes in Computer Science, vol. 8659. Springer, 2014, pp. 392–399.

[11] Y. Wang, L. Wen, Z. Yan, B. Sun, and J. Wang, "Discovering BPMN models with sub-processes and multi-instance markers," in *OTM Conferences*, ser. Lecture Notes in Computer Science, vol. 9415. Springer, 2015, pp. 185–201.

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

[13] F. Mannhardt, M. de Leoni, H. A. Reijers, W. M. P. van der Aalst, and P. J. Toussaint, "From low-level events to activities - A pattern-based approach," in *BPM*, ser. Lecture Notes in Computer Science, vol. 9850. Springer, 2016, pp. 125–141.

[14] 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 *SANER*. IEEE Computer Society, 2018, pp. 185–196.

[15] A. Adriansyah, B. F. van Dongen, and W. M. P. van der Aalst, "Conformance checking using cost-based fitness analysis," in *Enterprise Distributed Object Computing Conference (EDOC), 2011 15th IEEE International*. IEEE, 2011, pp. 55–64.

[16] J. Munoz-Gama and J. Carmona, "A fresh look at precision in process conformance," in *BPM*, ser. Lecture Notes in Computer Science, vol. 6336. Springer, 2010, pp. 211–226.

[17] N. Tax, B. Dalmas, N. Sidorova, W. M. P. van der Aalst, and S. Norre, "Interest-driven discovery of local process models," *Inf. Syst.*, vol. 77, pp. 105–117, 2018.

[18] A. Senderovich, M. Weidlich, A. Gal, A. Mandelbaum, S. Kadish, and C. A. Bunnell, "Discovery and validation of queueing networks in scheduled processes," in *International Conference on Advanced Information Systems Engineering*. Springer, 2015, pp. 417–433.

[19] S. J. van Zelst, F. Mannhardt, M. de Leoni, and A. Koschmider, "Event abstraction in process mining: literature review and taxonomy," *Granular Computing*, May 2020.

Identifying Candidate Routines for Robotic Process Automation from Unsegmented UI Logs

Volodymyr Leno*†, Adriano Augusto*, Marlon Dumas†, Marcello La Rosa*,
Fabrizio Maria Maggi‡†, Artem Polyvyanyy*
*University of Melbourne, Australia
{a.augusto, marcello.larosa, artem.polyvyanyy}@unimelb.edu.au
†University of Tartu, Estonia
{leno, marlon.dumas}@ut.ee
‡Free University of Bozen-Bolzano, Italy
maggi@inf.unibz.it

***Abstract*—Robotic Process Automation (RPA) is a technology to develop software bots that automate repetitive sequences of interactions between users and software applications (a.k.a. routines). To take full advantage of this technology, organizations need to identify and to scope their routines. This is a challenging endeavor in large organizations, as routines are usually not concentrated in a handful of processes, but rather scattered across the process landscape. Accordingly, the identification of routines from User Interaction (UI) logs has received significant attention. Existing approaches to this problem assume that the UI log is segmented, meaning that it consists of traces of a task that is presupposed to contain one or more routines. However, a UI log usually takes the form of a single unsegmented sequence of events. This paper presents an approach to discover candidate routines from unsegmented UI logs in the presence of noise, i.e. events within or between routine instances that do not belong to any routine. The approach is implemented as an open-source tool and evaluated using synthetic and real-life UI logs.**

***Index Terms*—Robotic process automation, robotic process mining, user interaction log.**

I. Introduction

Robotic Process Automation (RPA) allows organizations to improve their processes by automating repetitive sequences of interactions between a user and one or more software applications (a.k.a. routines). With this technology, it is possible to automate data entry, data transfer, and verification tasks, particularly when such tasks involve multiple applications. To exploit this technology, organizations need to identify routines that are prone to automation [1]. This can be achieved via interviews or manual observation of workers. In large organizations, however, this approach is not always cost-efficient as routines tend to be scattered across the process landscape. In this setting, manual routines identification efforts can be enhanced via automated methods, for example methods that extract frequent patterns from these User Interaction (UI) logs of working sessions of one or more workers [2].

Existing methods in this space [3]–[6] assume that the event log consists of a set of traces of a task that is presupposed to contain one or more routines. When the log is segmented, the identification of candidate routines boils down to discovering frequent sequential patterns from a collection of sequences, a problem for which a range of algorithms exist.

In practice, though, UI logs are not segmented. Instead, a recording of a working session consists of a single sequence of actions encompassing many instances of one or more routines, interspersed with other events that may not be part of any routine. Traditional approaches to sequential pattern mining, particularly those that are resilient to noise (irrelevant events) are not applicable to such unsegmented logs.

This paper addresses this gap by proposing a method to automatically split an unsegmented UI log into a set of segments, each representing a sequence of steps that appears repeatedly in the unsegmented UI log. Once the log is segmented, sequential pattern mining techniques are used to discover candidate routines. The method has been evaluated on synthetic and real-life UI logs in terms of its ability to rediscover routines contained in a log and in terms of scalability.

The paper is structured as follows. Section II provides the necessary background and an overview of related work. Section III describes the approach, while Section IV reports the results of the evaluation. Finally, Section V concludes the paper and spells out directions for future work.

II. Background and Related Work

The problem addressed by this paper can be framed in the broader context of Robotic Process Mining (RPM) [2]. RPM is a family of methods to discover repetitive routines performed by employees during their daily work, and to turn such routines into software scripts that emulate their execution. The first step in an RPM pipeline is to record the interactions between one or more workers and one or more software applications [7]. The recorded data is represented as a UI log – a sequence of user interactions (herein called UIs), such as selecting a cell in a spreadheet or editing a text field in a form. The UI log may be filtered to remove irrelevant UIs (e.g. misclicks). Next, it may be decomposed into segments. The discovered segments are scanned to identify routines that occur across these segments. Finally, the resulting routines are analysed to identify those that are automatable and to encode them as RPA scripts.

The problem of routines identification from UI logs in the context of RPM has attracted significant attention [7]. However, existing approaches for routines identification from UI logs either take as input a segmented UI log [3], [5], [6], [8], or they assume that the log can be trivially segmented by breaking it down at each point where one among a set of "start

978-1-7281-9832-3/20/$31.00 ©2020 IEEE
DOI 10.1109/ICPM49681.2020.00031

events" appears [4]. These start events, which act as delimiters between segments, need to be designated by the user.

Another technique for routines identification [1] attempts to identify candidate routines from textual documents – an approach that is suitable for earlier stages of routines identification and could be used to determine which processes or tasks could be recorded and analyzed in order to identify routines.

Below, we review the literature related to UI log segmentation and identification of routines from (segmented) logs.

A. UI Log Segmentation

Given a UI log, i.e. a sequence of UIs, segmentation consists in identifying non-overlapping subsequences of UIs, namely *segments*, such that each subsequence represents the execution of a task performed by an employee from start to end. In other words, segmentation searches for repetitive patterns in the UI log. In an ideal scenario, we would observe only one unique pattern (the task execution) repeated a finite number of times. However, in reality, this scenario is unlikely to materialise. Instead, it is reasonable to assume that an employee performing X-times the same task would do some mistakes or introduce variance in how the task is performed.

The problem of segmentation is similar to periodic pattern mining on time series. While several studies addressed the latter problem over the past decades [9], [10], most of them require information regarding the length of the pattern to discover, or assume a natural period to be available (e.g. hour, day, week). This makes the adaptation of such techniques to solve the problem of segmentation challenging, unless periodicity and pattern length are known a-priori.

Under the same class of problems, we find web session reconstruction [11], whose goal is to identify the beginning and the end of web navigation sessions in server log data, e.g. streams of clicks and web pages navigation [11]. Methods for session reconstruction are usually based on heuristics that rely either on IP addresses or on time intervals between events. The former approach is not applicable in our context (routines in UI logs cannot be segmented based on IP addresses), while the latter approach assumes that users make breaks in-between two consecutive segments – in our case, two routine instances.

Lastly, segmentation also relates to the problem of correlation of event logs for process mining. In such logs, each event should normally include an identifier of a process instance (case identifier), a timestamp, an activity label, and possibly other attributes. When the events in an event log do not contain explicit case identifiers, they are said to be uncorrelated. Various methods have been proposed to extract correlated event logs from uncorrelated ones. However, existing methods in this field either assume that a process model is given as input [12] or that the underlying process is acyclic [13]. Both of these assumptions are unrealistic in our setting: a process model is not available since we are exactly trying to extract that information from the log (by identifying the routines), and a routine may contain repetitions.

B. Routines Identification

Once the UI log is segmented, the segments are scanned to identify routines. Dev and Liu [3] have noted that the problem of routines identification from (segmented) UI logs can be mapped to that of frequent pattern mining, a well-known problem in the field of data mining [14]. Indeed, the goal of routines identification is to identify repetitive (frequent) sequences of interactions, which can be represented as symbols. In the literature, several algorithms are available to mine frequent patterns from sequences of symbols. Depending on their output, we can distinguish two types of frequent pattern mining algorithms: those that discover only exact patterns [15], [16] (hence vulnerable to noise) and those that allow frequent patterns to have gaps within the sequence of symbols [17], [18] (hence noise-resilient). Depending on their input, we can distinguish between algorithms that operate on a collection of sequences of symbols and those that discover frequent patterns from a single long sequence of symbols [16]. The former algorithms can be applied to segmented UI logs while the latter can be applied directly to unsegmented ones. However, techniques that identify patterns from a single sequence of symbols only scale up when identifying exact patterns.

III. Approach

In this section, we describe our approach for identifying candidate routines in UI logs. As input, the approach takes a preprocessed and normalized UI log and outputs a set of candidate routines. The approach follows the RPM pipeline [2], and consists of two macro steps. First, the normalized UI log is decomposed into segments. Then, candidate routines are identified by mining frequent sequential patterns from the segments. In this paper, we refer to these two macro steps as *segmentation* and *candidate routines identification*, respectively. The approach is summarized in Fig. 1. Next, we describe the steps in detail, including the required UI log preprocessing and normalization.

A. UI Log Preprocessing and Normalization

Before we proceed, we give some formal definitions necessary to support the subsequent discussions.

*Definition 1 (**User Interaction (UI)**):* A *user interaction* (UI) is a tuple $u = (t, \tau, P, Z, \phi)$, where: t is a UI timestamp; τ is a UI type (e.g. click button, copy cell); P_τ is a set of UI parameters (e.g. button name, worksheet name, url, etc.); Z is a set of UI parameters values; and $\phi : P_\tau \rightarrow Z$ is a function that maps UI parameters onto values.

*Definition 2 (**UI Log**):* A UI Log Σ is a sequence of user interactions $\Sigma = \langle u_1, u_2, \ldots, u_n \rangle$, ordered by timestamp, i.e. $u_{i|t} < u_{j|t}$ for any $i, j \mid 1 \leq i < j \leq n$. In the remainder of this paper, we also call a UI log simply log.

Ideally, the UIs recorded in a UI log should capture only the execution of the task under recording. However, a log often contains also UIs that do not bring any value to the recorded task, and that should not have been executed in the first place. Some common examples of such UI *noise* include: an employee replying to incoming emails and/or browsing the web while executing a different task; or an employee committing mistakes, e.g. filling a text field with an incorrect value, or copying an incorrect text or cell in a spreadsheet. To reduce the impact of noise on routines identification, we preprocess the log. The preprocessing we apply consists

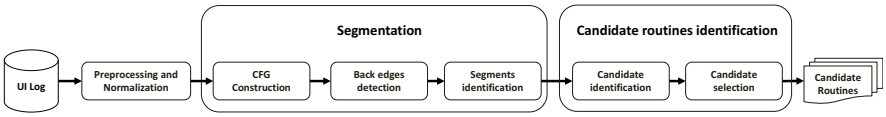


Fig. 1: Outline of the proposed approach

in identifying and removing redundant UIs that are clearly overwritten by successive UIs, such as the case of a double *CTRL+C* performed in sequence on different text fields. To identify such patterns of UIs, we rely on regular expressions by applying the methods described in [19].

After the preprocessing, the vast majority of UIs in the log are unique, because they differ by their payload. Even UIs capturing the same action within the same task execution (or different task executions) appear to be different. To discover each task execution (from start to end) recorded in the UI log, we need to detect all the UIs that, even having different payloads, correspond to the same action within the task execution.

To do so, we need to introduce the concepts of *UI data parameter*, *UI context parameter*, and *normalized UI*. Given a UI, its parameters can be divided into two types: *data parameter* and *context parameter*. The *data parameters* store the data values that are used during the execution of a task, e.g. the value of text fields or copied content. The *data parameters* usually have different values per task execution. By contrast, the *context parameters* capture where the UI was performed, e.g. the application and the location within the application. The *context parameters* are likely to have always the same values even in different task executions. For example, a UI of type *copy* includes the following parameters: *target-application* (e.g. the browser), *user*, *element id* (e.g. the id of a text field in the browser), *copied content*. Here, *target-application* and *element id* are context parameters, while *copied content* is a data parameter. Naturally, different type of UIs are characterized by different context parameters, e.g. a UI in a spreadsheet has the following context parameters: *spreadsheet name* and *cell location*.[1]

*Definition 3 (**Normalized UI**):* Given a UI $u = (t, \tau, P_\tau, Z, \phi)$, we define its normalization as $\bar{u} = (t, \tau, \bar{P}_\tau, \bar{Z}, \phi)$; where $\bar{Z} \subseteq Z$ contains only the values of the parameters in $\bar{P}$, where $\bar{P}$ is a set of context parameters. Given two normalized UIs, $u_1 = (t_1, \tau, P_\tau, \bar{Z}_1, \phi_1), u_2 = (t_2, \tau, P_\tau, \bar{Z}_2, \phi_2)$, the equality relation $u_1 = u_2$ holds iff $\forall p \in P_\tau \Rightarrow \phi_1(p) = \phi_2(p)$.

Given a UI log $\Sigma = \langle u_1, u_2, \ldots, u_n \rangle$, we normalize it by normalizing all the recorded UIs. The resulting normalized UI log is $\bar{\Sigma} = \langle \bar{u}_1, \bar{u}_2, \ldots, \bar{u}_n \rangle$. Table I and II show, respectively, a small fragment of a UI log and its normalized version. Intuitively, in a normalized UI log, the chances that two executions of the same routine have the same sequence (or set) of normalized UIs are high, because normalized UIs only have context parameters. In the next steps, we leverage such a characteristic of the normalized UI log to identify its segments (i.e. start and end of each executed task), and the routine(s) within the segments.

[1]The context parameters are selected by the domain expert.

	UI Timestamp	UI Type	UI Parameters and Values		
			P_1: Application	P_2: Element Label	P_3: Element Value
1	2019-03-03T19:02:18	Click button	Web	New Record	–
2	2019-03-03T19:02:23	Edit field	Web	Full Name	Albert Rauf
3	2019-03-03T19:02:27	Edit field	Web	Date	11-04-1986
4	2019-03-03T19:02:39	Edit field	Web	Phone	+ 61 043 512 4834
5	2019-03-03T19:02:47	Click button	Web	Submit	–
6	2019-03-03T19:02:58	Click button	Web	New Record	–
7	2019-03-03T19:03:13	Edit field	Web	Date	20-06-1987
8	2019-03-03T19:03:24	Edit field	Web	Phone	+61 519 790 1066
9	2019-03-03T19:03:43	Edit field	Web	Full Name	Audrey Backer
10	2019-03-03T19:04:10	Click button	Web	Submit	–

TABLE I: Fragment of a UI log

	UI Timestamp	UI Type	UI Parameters and Values	
			P_1: Application	P_2: Element Label
1	2019-03-03T19:02:18	Click button	Web	New Record
2	2019-03-03T19:02:23	Edit field	Web	Full Name
3	2019-03-03T19:02:27	Edit field	Web	Date
4	2019-03-03T19:02:39	Edit field	Web	Phone
5	2019-03-03T19:02:47	Click button	Web	Submit
6	2019-03-03T19:02:58	Click button	Web	New Record
7	2019-03-03T19:03:13	Edit field	Web	Date
8	2019-03-03T19:03:24	Edit field	Web	Phone
9	2019-03-03T19:03:43	Edit field	Web	Full Name
10	2019-03-03T19:04:10	Click button	Web	Submit

TABLE II: Normalized UI log

B. Segmentation

Before describing in detail the segmentation step, we formally define the necessary cornerstone concepts.

*Definition 4 (**Directly-follows relation**):* Let $\bar{\Sigma} = \langle \bar{u}_1, \bar{u}_2, \ldots, \bar{u}_n \rangle$ be a normalized UI log. Given two normalized UIs, $\bar{u}_x, \bar{u}_y \in \bar{\Sigma}$, we say that $\bar{u}_y$ directly-follows $\bar{u}_x$, i.e. $\bar{u}_x \rightsquigarrow \bar{u}_y$, iff $\bar{u}_{x|t} < \bar{u}_{y|t} \wedge \nexists \bar{u}_z \in \Sigma \mid \bar{u}_{x|t} \geq \bar{u}_{z|t} \geq \bar{u}_{y|t}$.

*Definition 5 (**Control-Flow Graph (CFG)**):* Given a normalized UI log, $\bar{\Sigma} = \langle \bar{u_1}, \bar{u_2}, \ldots, \bar{u_n} \rangle$, let $\bar{A}$ be the set of normalized UIs in $\bar{\Sigma}$. A Control-Flow Graph (CFG) is a tuple $G = (V, E, \hat{v}, \hat{e})$, where: V is the set of vertices of the graph, each vertex maps one normalized UI in $\bar{A}$; $E \subseteq V \times V$ is the set of edges of the graph, and each $(v_i, v_j) \in E$ represents a directly-follows relation between the UIs mapped by v_i and v_j; $\hat{v}$ is the graph *entry vertex*, such that $\forall v \in V \nexists (v, \hat{v}) \in E \wedge \nexists (\hat{v}, v) \in E$; while $\hat{e} = (\hat{v}, v_0)$ is the graph *entry edge*, such that v_0 maps $\bar{u_1}$. We note that $\hat{v} \notin V$, and $\hat{e} \notin E$, since they are artificial elements of the graph.

*Definition 6 (**CFG Path**):* Given a CFG $G = (V, E, \hat{v}, \hat{e})$, a CFG path is a sequence of vertices $p_{v_1, v_k} = \langle v_1, \ldots, v_k \rangle$ such that for each $i \in [1, k-1] \Rightarrow v_i \in V \cup \{\hat{v}\} \wedge \exists (v_i, v_{i+1}) \in E \cup \{\hat{e}\}$.

*Definition 7 (**Strongly Connected Component (SCC)**):* Given a graph $G = (V, E, \hat{v}, \hat{e})$, a strongly connected component (SCC) of G is a pair $\delta = (\bar{V}, \bar{E})$, where $\bar{V} = \{v_1, v_2, \ldots, v_m\} \subseteq V$ and $\bar{E} = \{e_1, e_2, \ldots, e_k\} \subseteq E$ such that $\forall v_i, v_j \in \bar{V} \exists p_{v_i, v_j} \mid \forall v \in p \Rightarrow v \in \bar{V}$. Given an SCC $\delta = (\bar{V}, \bar{E})$, we say that δ is

non-trivial iff $|\bar{V}| > 1$. Given a graph G, Δ_G denotes the set of all the non-trivial SCCs in G.

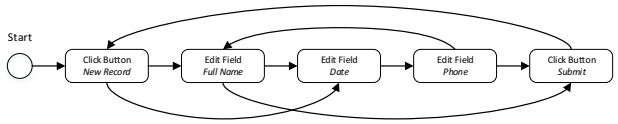


Fig. 2: Example of a Control-Flow Graph

The segmentation step starts with the construction of the CFG of the input normalized UI log. It is common that a CFG contains loops, since a loop represents the start of a new execution of the task recorded in the UI log. Indeed, in an ideal scenario, once a task execution ends with a certain UI (a vertex in the CFG), the next UI (i.e. the first UI of the next task execution) should already be mapped in the CFG and a loop will be generated. In such case, all the vertices contained in the loop represent the corresponding UIs that belong to the task. If several different tasks are recorded in the UI log, the graph would contain multiple disjoint loops, while if a task has repetitive subtasks there would be nested loops. Fig. 2 shows the CFG generated from the normalized UI log captured in Table II.

After the CFG is generated, we turn our attention to the identification of its back-edges, which can be detected by analysing the SCCs of the CFG, as described in Algorithm 1 and Algorithm 2. Given a CFG $G = (V, E, \hat{v}, \hat{e})$, we first build its dominator tree Θ (Algorithm 1, line 2), which captures domination relations between the vertices of the CFG. Fig. 3 shows the dominator tree of the CFG in Fig. 2.

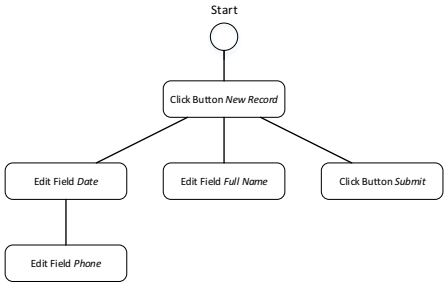


Fig. 3: Dominator tree

Then, we discover the set of all non-trivial SCCs (Δ_G) by applying the Kosaraju's algorithm [20] and removing the trivial SCCs (Algorithm 1, line 3). For each $\delta = (\bar{V}, \bar{E}) \in \Delta_G$, we discover its *header* using the dominator tree (Algorithm 2, line 1). The header of a δ is a special vertex $\hat{h} \in \bar{V}$, such that $\forall p_{\hat{v},v} \mid v \in \bar{V} \Rightarrow \hat{h} \in p_{\hat{v},v}$, i.e. the *header* $\hat{h}$ is the SCC vertex that dominates all the other SCC vertices. Once we have $\hat{h}$, we can identify the back-edges as $(v, \hat{h})$ with $v \in \bar{V}$ (line 3). Finally, the identified back-edges are stored and removed (lines 4 and 5) in order to look for nested SCCs and their back-edges by recursively executing Algorithm 2 (line 11), until no more SCCs and back-edges are found. However, if we detect an SCC that does not have a header vertex (i.e. the SCC is irreducible), we cannot identify the SCC back-edges. In such a case, we collect via a depth-first search of the CFG the edges $(v_x, v_y) \in \bar{E}$ such that v_y is topologically deeper than v_x - we call these edges *loop-edges* of the SCC (line 7). Then, out of all the loop-edges, we store (and remove from the SCC) the one having target and source connected by the longest *simple path* entirely contained within the SCC (lines 8 to 9).

Algorithm 1: Detect Back-edges

```
   input  : CFG G
   output : Back-edges Set B
1  B ← ∅;
2  Dominator Tree Θ ← computeDominatorTree(G);
3  Set Δ_G ← findSCCs(G);
4  foreach δ ∈ Δ_G do AnalyseSCC(δ, Θ, B);
5  return B;
```

Algorithm 2: Analyse SCC

```
   input  : SCC δ = (V̄, Ē), Dominator Tree Θ, Back-edges Set B
1  Header ĥ ← findHeader(δ, Θ);
2  if ĥ ≠ null then
3      Set I ← getIncomingEdges(δ, ĥ);
4      B ← B ∪ I;
5      Ē ← Ē \ I;
6  else
7      Set L ← findLoopEdges(δ);
8      Edge e ← getTheDeepestEdge(δ, L);
9      remove e from Ē;
10 Set Δ_δ ← findSCCs(δ);
11 foreach γ ∈ Δ_δ do AnalyseSCC(γ, Θ, B);
```

Given the CFG presented in Fig. 2 and its corresponding dominator tree (see Fig. 3), we immediately identify the SCC that consists of all the vertices except the *entry vertex*. Then, by applying Algorithm 2, we identify: the SCC header – *Click Button [New Record]*; and the only back-edge – (*Click Button [Submit]*, *Click Button [New Record]*), which we save and remove from the SCC. After the removal of this back-edge, we identify the nested SCC that contains all the three *Edit Field* UIs. Note that this second SCC does not have a header because it is irreducible, due to its multiple entries (*Edit Field [Full Name]* and *Edit Field [Date]*). However, by applying the depth-first search, we identify as candidate loop-edge for removal: (*Edit Field [Phone]*, *Edit Field [Full Name]*). After we remove this edge from the CFG, no SCCs are left so that Algorithm 2 stops its recursion.

At this point, we collected all the back-edges of the CFG, and we can leverage this information to start segmenting the UI log. We do so via Algorithm 3. First, we retrieve all the targets and sources of all the back-edges in the CFG and collect their corresponding UIs (lines 2 and 3). Each UI that is the target of a back-edge is an eligible segment starting point (hereinafter, *segment-start UI*), since a back-edge conceptually captures the end of a task execution, therefore its target represents the first UI of the next task execution. Following the same reasoning, each UI that is source of a back-edge is an eligible segment ending point (hereinafter, *segment-end UI*). Then, we sequentially scan all the UIs in the UI log (line 7). When we encounter a segment-start UI (line 9), and we are not already within a segment (line 10), we create a new segment (s, a list of UIs), we append the segment-start UI ($\bar{u}$), and we store it in order to match it with the correct segment-end UI (lines 11

Algorithm 3: Identify Segments

```
   input  : Normalised UI log Σ̂, Back-edges Set B
   output : Segments List Ψ
1  Set Ψ ← ∅;
2  Set T ← getTargets(B);
3  Set S ← getSources(B);
4  Boolean WithinSegment ← false;
5  Normalised UI u_0 ← null;
6  List s ← null;
7  for i ∈ [1, size(Σ)] do
8      Normalised UI ū ← getUI(Σ̂, i);
9      if ū ∈ T then
10         if WithinSegment = false then
11             s ← new List;
12             append ū to s;
13             u_0 ← ū;
14             WithinSegment ← true;
15         else
16             append ū to s;
17     else
18         if WithinSegment = true then
19             append ū to s;
20             if ū ∈ S ∧ (ū, u_0) ∈ B then
21                 add s to Ψ;
22                 WithinSegment ← false;
23 return Ψ;
```

to 14). Our strategy to detect segments in the UI log is driven by the following underlying assumption: a specific segment-end UI will be followed by the same segment-start UI, so that we can match segment-end and segment-start UIs exploiting back-edge's sources and targets (respectively). If the UI is not a segment-start (line 17), we check if we are within a segment (line 18) and, if not, we discard the UI, assuming that it is noise since it fell between the previous segment-end UI and the next segment-start UI. Otherwise, we append the UI to the current segment and we check if the UI is a segment-end matching the current segment-start UI (line 20). If that is the case, we reached the end of the segment and we add it to the set of segments (line 21), otherwise, we continue reading the segment.

Table III shows the segment-start and segment-end UIs, highlighted respectively in green and red, which also ideally delimits the two segments within the normalized UI log.

	UI Timestamp	UI Type	UI Parameters and Values	
			P_1: Application	P_2: Element Label
1	2019-03-03T19:02:18	Click button	Web	New Record
2	2019-03-03T19:02:23	Edit field	Web	Full Name
3	2019-03-03T19:02:27	Edit field	Web	Date
4	2019-03-03T19:02:39	Edit field	Web	Phone
5	2019-03-03T19:02:47	Click button	Web	Submit
6	2019-03-03T19:02:58	Click button	Web	New Record
7	2019-03-03T19:03:13	Edit field	Web	Date
8	2019-03-03T19:03:24	Edit field	Web	Phone
9	2019-03-03T19:03:43	Edit field	Web	Full Name
10	2019-03-03T19:04:10	Click button	Web	Submit

TABLE III: Segments identification

C. Candidate routines identification

The candidate routines identification step is based on the CloFast sequence mining algorithm [18]. To embed CloFast in our approach, we have to define the structure of the sequential patterns we want to identify. In this paper, we define a *sequential pattern* within a UI log as a sequence of normalized UIs occurring always in the same order in different segments, yet allowing gaps between the UIs belonging to the pattern. For example, if we consider the following three segments: $\langle u_1, u_y, u_2, u_3 \rangle$, $\langle u_1, u_2, u_x, u_3 \rangle$, and $\langle u_1, u_x, u_2, u_3 \rangle$; they all contain the same sequential pattern that is $\langle u_1, u_2, u_3 \rangle$. Furthermore, we define the *support* of a sequential pattern as the ratio of its occurrences and the total number of segments, and we refer to *closed* patterns and *frequent* patterns (relatively to an input threshold) as they are known in the literature. In particular, a frequent pattern is a pattern that appears in at least a number of sequences indicated by the threshold, while a closed pattern is a pattern that is not included in another pattern having exactly the same support. By applying CloFast to the set of the UI log segments, we discover all the *frequent closed* sequential patterns.

Some of these patterns may be *overlapping*, which (in this context) means they were discovered from the same portion of a segment and share some UIs. An example of overlapping patterns is the following, given three segments: $\langle u_1, u_y, u_2, u_3, u_x, u_4 \rangle$, $\langle u_1, u_y, u_2, u_x, u_3, u_4 \rangle$, and $\langle u_1, u_x, u_2, u_3, u_4 \rangle$; $\langle u_1, u_2, u_3, u_4 \rangle$ and $\langle u_1, u_x, u_4 \rangle$ are sequential patterns, but they overlap due to the shared UIs (u_1 and u_4). In practice, each UI belongs to only one routine, therefore, we are interested in discovering only non-overlapping patterns. For this purpose, we implemented an optimization that we use on top of CloFast. Given the set of patterns discovered by CloFast, we rank them by a pattern quality criterion (e.g. length, frequency), and we select the best pattern (i.e. the top rank one). Then, all its occurrences are removed from the segments, and we search again for frequent closed patterns performing the same procedure until there are no frequent closed patterns left.

In our approach, we integrated four pattern quality criteria to select the candidate routines: pattern frequency, pattern length, pattern coverage, and pattern cohesion score [3]. Pattern frequency considers how many times the pattern was observed in different segments. Pattern length considers the length of the patterns. Pattern coverage considers the percentage of the log that is covered by all the pattern occurrences. Pattern cohesion score considers the level of adjacency of the elements inside a pattern and is calculated as the difference between the pattern length and the median number of gaps between its elements. In other words, cohesion prioritizes the patterns whose UIs appear consecutively without (or with few) gaps while taking into account also the pattern length. In the next section, we compare these ranking criteria and discuss the benefits of using one or another.

IV. Evaluation

We implemented our approach as an open-source Java command-line application.[2] Our goal is threefold. First, we assess to what extent our approach can rediscover routines that are known to be recorded in the input UI logs. Second, we analyze how the use of different candidate routines selection criteria such as frequency and cohesion impact on the quality

[2]Available at https://github.com/volodymyrLeno/RPM_Segmentator

of the discovered routines. Last, we assess the efficiency and effectiveness of our approach when applied to real-life UI logs. Accordingly, we define the following research questions:

- **RQ1.** Does the approach rediscover routines that are known to exist in a UI log?
- **RQ2.** How do the candidate routines selection criteria affect the quality of the discovered routines?
- **RQ3.** Is the approach applicable in real-life settings, in terms of both efficiency and effectiveness?

A. Datasets

To answer our research questions, we rely on a dataset of thirteen UI logs, which can be divided into three subgroups: artificial logs, real-life logs recorded in a supervised environment, and real-life logs recorded in an unsupervised environment.[3] Table IV shows the logs characteristics.

UI Log	# Routine Variants	# Task Traces	# Actions	# Actions per trace (Avg.)
CPN1	1	100	1400	14.000
CPN2	3	1000	14804	14.804
CPN3	7	1000	14583	14.583
CPN4	4	100	1400	14.000
CPN5	36	1000	8775	8.775
CPN6	2	1000	9998	9.998
CPN7	14	1500	14950	9.967
CPN8	15	1500	17582	11.721
CPN9	38	2000	28358	14.179
Student Records (SR)	2	50	1539	30.780
Reimbursement (RT)	1	50	3114	62.280
Scholarships 1 (S1)		-	693	
Scholarships 2 (S2)		-	509	

TABLE IV: UI logs characteristics

The artificial logs (CPN1–CPN9) were generated from Colored Petri Nets (CPNs) [5]. The CPNs have increasing complexity, from low (CPN1) to high (CPN9). These logs are originally noise-free and segmented. We removed the segment identifiers to produce unsegmented logs.

The *Student Records* (SR) and *Reimbursement* (RT) logs record the simulation of real-life scenarios. The SR log simulates the task of transferring students' data from a spreadsheet to a Web form. The RT log simulates the task of filling reimbursement requests with data provided by a claimant. Each log contains fifty recordings of the corresponding task executed by one of the authors, who followed strict guidelines on how to perform the task. These logs contain little noise, which only accounts for user mistakes, such as filling the form with an incorrect value and performing additional actions to fix the mistake. For both logs, we know how the underlying task was executed, and we treat such information as ground truth when evaluating our approach. Additionally, we created four more logs ($SRRT_+$, $RTSR_+$, $SRRT_\parallel$, $RTSR_\parallel$) by combining SR and RT. $SRRT_+$ and $RTSR_+$ capture the scenario where the user first completes all the instances of one task and then moves to the other task. These logs were generated by concatenating SR and RT. $SRRT_\parallel$ and $RTSR_\parallel$ capture the scenario where the user is working simultaneously on two tasks. To simulate such behavior, we interleaved the segments of SR with those of RT.

[3]The real-life logs were recorded with the Action Logger tool [7]. All the logs are available at https://doi.org/10.6084/m9.figshare.12543587

Finally, the *Scholarships* logs (S1 and S2) were recorded by two employees of the University of Melbourne who performed the same task. The logs record the task of processing scholarship applications for international and domestic students. The task mainly consists of students data manipulation with transfers between spreadsheets and Web pages. Compared to the other logs, we have no a-priori knowledge of how to perform the task in the *Scholarships* logs (no ground truth). Also, when recording the UI logs, the University employees were not instructed to perform their task in a specific manner, i.e. they were left free to perform the task as they would normally do when unrecorded.

B. Setup

To answer RQ1 and RQ2, we analyzed the quality of the segmentation and that of the discovered routines, using the first 15 logs described above (CPN1 to CPN9, SR, RT, $SRRT_+$, $RTSR_+$, $SRRT_\parallel$, $RTSR_\parallel$) against the four candidate routines selection criteria in Section III-C, i.e. frequency, length, coverage and cohesion. To assess the quality of the segmentation, we use the *normalized* Levenshtein Edit Distance (LED), where a segment and its normalized UIs represent the string and its characters, respectively. Precisely, for each discovered segment, we collect all the ground truth segments that have at least one shared UI with the discovered segment, calculate the LED between the discovered segment and the ground truth segments and assign the minimum LED to the discovered segment as its quality score. Finally, we assess the overall quality of the segmentation as the average of the LEDs of each discovered segment.

The quality of the discovered routines is measured with the Jaccard Coefficient (JC), which captures the level of similarity between discovered and ground truth routines in a less strict manner compared to LED. In fact, the JC does not penalize the order of the UIs in a routine. This follows from the assumption that a routine could be executed performing some actions in different order, and the ordering should not be penalized. The JC between two routines is the ratio $\frac{n}{m}$, where n is the number of UIs that are contained in both routines, while m is the total number of UIs in each of the two routines (i.e. the sum of the lengths of the two routines). Given the set of discovered routines and the set of ground truth routines, for each discovered routine, we compute its JC with all the ground truth routines and assign the maximum JC to the discovered routine as its quality score. Finally, we assess the overall quality of the discovered routines as the average of the JC of each discovered routine. As the ground truth, we used the segments of the artificial logs and the guidelines given to the author who performed the tasks in SR and RT.

However, we cannot rely on the JC alone to assess the quality of the discovered routines, as this measure does not consider the routines we may have missed in the discovery. Thus, we also measure the total coverage to quantify how much log behavior is captured by the discovered routines. We would like to reach high coverage with as few routines as possible. Thus, we prioritize long routines over short ones by measuring the average routine length alongside the coverage.

To answer RQ3, we tested our approach on the S1 and S2 logs and qualitatively assessed the results with the help of the employees who performed the task. Specifically, we asked them to compare the rediscovered routines and the actions (i.e. UIs) they performed while recording.

All experiments were conducted on a Windows 10 laptop with an Intel Core i5-5200U CPU 2.20 GHz and 16GB RAM.

C. Results

Table V shows the results of the segmentation. As we can see, the LED for all the CPN logs is 0.0, highlighting that all the segments were discovered correctly. On the other hand, the segments discovered from the SR, RT, $SRRT_{+}$, $RTSR_{+}$, $SRRT_{\|}$, and $RTSR_{\|}$ logs slightly differ from the original ones. The main difference between the CPN logs and those recorded in a controlled environment is that the former contain routines having always the same starting UI, while the latter contain routines with several different starting UIs.

We identified the correct number of segments from all the logs except $SRRT_{\|}$ and $RTSR_{\|}$, where we could not discern the ending UI of the routine belonging to one task and the starting UI of the routine belonging to the other task, consequently merging the two routines and discovering only half of the total number of segments (50 out of 100). From the table we can also see that the time performance of our approach is reasonable, with maximum execution time of 3.6 seconds.

UI Log	# Original Segments	# Discovered Segments	LED (avg)	Exec. Time
CPN1	100	100	0.000	0.571
CPN2	1000	1000	0.000	1.705
CPN3	1000	1000	0.000	0.835
CPN4	100	100	0.000	0.461
CPN5	1000	1000	0.000	1.025
CPN6	1000	1000	0.000	0.707
CPN7	1500	1500	0.000	1.566
CPN8	1500	1500	0.000	1.596
CPN9	2000	2000	0.000	3.649
SR	50	50	0.059	0.714
RT	50	50	0.095	1.662
$SRRT_{+}$	100	100	0.078	2.424
$RTSR_{+}$	100	100	0.078	2.221
$SRRT_{\|}$	100	50	0.331	2.296
$RTSR_{\|}$	100	50	0.331	2.536

TABLE V: Segmentation results

Table VI shows the quality of the discovered routines for each selection criterion, when setting 0.1 as minimum support threshold of CloFast. The results highlight that, overall, the routines with the highest JC and the longest length are those discovered using cohesion as selection criterion, followed closely by those discovered using length as the criterion. Even though using these criteria we do not always achieve the highest coverage, the coverage scores are very high, i.e. above 0.90 for all the logs except CPN5.

Following these results, we decided to use cohesion as the selection criterion to discover the routines from the *scholarships* logs. From the S1 log we discovered five routines variants. The first routine variant consists in manually adding graduate research student applications to the student record in the information system of the university. The application is then assessed, and the student is notified about the outcome. The second routine variant consists in lodging a ticket to verify possible duplicate applications. When a new application is entered in the information system and its data matches an existing application, the new application is temporarily put on hold, and the employee fills in and lodges a ticket to investigate the duplicate. The remaining three routine variants represent exceptional cases, where the employee executed either the first or the second variant in a different manner (i.e. by altering the order of the actions or with overlapping routines executions). To assess the results, we showed the discovered routine variant to the employee of the University of Melbourne who recorded the S1 log, and they confirmed that the discovered routines correctly capture their task executions. Also, they confirmed that the last three routine variants are alternative executions of the first routine variant.[4]

UI Logs	Selection Criterion	# Discovered Routines	Routine Length	Total Coverage	JC	Exec. Time
CPN1	Frequency	1	14.00	1.00	1.000	2.643
	Length	1	14.00	1.00	1.000	1.553
	Coverage	1	14.00	1.00	1.000	3.702
	Cohesion	1	14.00	1.00	1.000	**1.530**
CPN2	Frequency	3	6.33	**0.99**	0.452	3.908
	Length	2	**14.50**	0.95	**1.000**	4.789
	Coverage	2	14.00	**0.99**	0.964	**3.166**
	Cohesion	2	**14.50**	0.95	**1.000**	3.730
CPN3	Frequency	4	5.75	0.95	0.511	4.682
	Length	3	**14.33**	0.93	**1.000**	4.324
	Coverage	3	9.67	**0.96**	0.833	**3.940**
	Cohesion	3	**14.33**	0.93	**1.000**	6.237
CPN4	Frequency	1	12.00	0.86	0.857	3.452
	Length	2	**14.00**	**1.00**	**1.000**	**2.005**
	Coverage	1	13.00	0.93	0.929	3.351
	Cohesion	2	**14.00**	**1.00**	**1.000**	3.655
CPN5	Frequency	6	1.67	**0.86**	0.206	**6.418**
	Length	7	7.29	0.83	0.849	9.715
	Coverage	4	3.75	0.80	0.462	6.587
	Cohesion	8	**7.5**	**0.86**	**0.910**	18.206
CPN6	Frequency	3	4.67	1.00	0.485	4.250
	Length	2	**10.00**	1.00	**1.000**	2.924
	Coverage	3	4.67	1.00	0.485	**2.483**
	Cohesion	2	**10.00**	1.00	**1.000**	4.678
CPN7	Frequency	7	2.43	0.91	0.257	10.118
	Length	7	**9.57**	0.88	**0.986**	8.957
	Coverage	6	3.67	0.91	0.385	**7.203**
	Cohesion	7	9.43	**0.93**	0.971	11.983
CPN8	Frequency	5	4.20	0.75	0.337	11.801
	Length	6	**10.67**	**0.91**	**0.967**	9.070
	Coverage	5	7.60	0.89	0.618	**7.354**
	Cohesion	5	**10.67**	**0.91**	**0.967**	11.250
CPN9	Frequency	5	5.20	0.82	0.401	13.784
	Length	6	**14.67**	**0.95**	**1.000**	**8.265**
	Coverage	5	6.60	0.88	0.511	8.603
	Cohesion	6	**14.67**	**0.95**	**1.000**	13.943
SR	Frequency	3	10.00	0.96	0.356	3.883
	Length	3	**28.33**	**0.98**	**0.942**	**2.592**
	Coverage	2	15.50	0.96	0.532	2.635
	Cohesion	3	**28.33**	**0.98**	**0.942**	3.252
RT	Frequency	3	18.67	0.90	0.290	**4.63**
	Length	3	**56.33**	**0.96**	**0.829**	5.215
	Coverage	2	30.50	0.45	0.446	4.709
	Cohesion	3	**56.33**	**0.96**	**0.829**	6.585
$SRRT_{+}$	Frequency	5	16.80	0.90	0.374	13.826
	Length	4	**45.25**	**0.91**	**0.929**	**7.362**
	Coverage	2	42.50	0.86	0.921	8.177
	Cohesion	4	**45.25**	**0.91**	**0.929**	10.728
$RTSR_{+}$	Frequency	5	16.80	0.90	0.374	12.176
	Length	4	**45.25**	**0.91**	**0.929**	12.477
	Coverage	2	42.50	0.86	0.921	**9.085**
	Cohesion	4	**45.25**	**0.91**	**0.929**	14.675
$SRRT_{\|}$	Frequency	3	28.00	0.90	0.313	**13.905**
	Length	5	**86.40**	**0.96**	**0.580**	28.259
	Coverage	3	55.00	0.95	0.391	14.894
	Cohesion	5	**86.40**	**0.96**	**0.580**	37.277
$RTSR_{\|}$	Frequency	3	28.00	0.90	0.313	12.428
	Length	4	**89.50**	0.90	**0.600**	23.903
	Coverage	3	55.00	**0.95**	0.391	**10.838**
	Cohesion	4	**89.50**	0.90	**0.600**	38.657

TABLE VI: Quality of the discovered routines

While the results from the S1 log were positive, our

[4]Detailed results at https://doi.org/10.6084/m9.figshare.12543587

approach could not discover any correct routine from the S2 log. By analyzing the results, we found out that the employee worked with multiple worksheets at the same time, frequently switching between them for visualization purposes only. Such behavior recorded in the log negatively affects the construction of the CFG and its domination tree, ultimately leading to the discovery of incorrect segments and routines. This also had an impact on the execution time, indeed, while it took only 41.7 seconds to discover the routines from the S1 log, it took 426.3 seconds to discover the routines from the S2 log.

D. Limitations

Our approach relies on information recorded in a UI log to identify segments and discover routines. Thus, its effectiveness is correlated with data quality. Since a UI log is fine-grained, deviations occurring during the routine execution affect the effectiveness of our approach. In our evaluation, we observed this phenomenon to varying degrees when dealing with real-life logs. In practice, the approach can identify correct routines only if they are observed frequently in the UI log. Recurring noise affects the accuracy of the results (see the S2 log).

The approach discovers multiple variants of the same routine when the UIs of a routine occur in different orders. Post-processing the results could be beneficial in order to cluster similar routines. Further, the approach is designed for logs that capture consecutive routine executions. In practice, routine instances may sometimes overlap (like in the S2 log).

Finally, while our approach is robust against routine executions with multiple ends, it is sensitive to multiple starts. Ideally, all routine executions should start with the same UI, unless different starts are recorded in batch (e.g. first only routines with a start, then routines with another start, etc.). In general, our approach can handle logs containing multiple different routines, provided that each routine does not share any UIs with other routines, except their start UIs.

V. Conclusion and Future Work

This paper presented an approach to automatically identify routines from unsegmented UI logs. The approach starts by decomposing the UI log into segments corresponding to paths within the connected components of a Control-Flow Graph derived from the log. Once the log is segmented, a noise-resilient sequential pattern mining technique is used to extract frequent patterns. The patterns are then ranked according to four quality criteria: frequency, length, coverage, and cohesion.

The approach has been implemented as an open-source tool and evaluated using synthetic and real-life logs. The evaluation shows that the approach can rediscover routines injected into a synthetic log, and that it discovers relevant routines in real-life logs. The execution times range from seconds to a few dozen seconds even for logs with tens of thousands of interactions.

As future work, we aim at addressing the limitations discussed above. We plan to add a post-processing step to group multiple routine variants and to discover an aggregated model thereof. This could be achieved by clustering the patterns and merging them into high-level models or by adapting a local process mining technique [21]. We plan to design more sophisticated segmentation techniques to better handle mixtures of multiple routines. Finally, the approach identifies routines from a control-flow perspective insofar as it manipulates sequences of interactions, without considering their data payload. Therefore, we plan to complement the proposed approach with an approach to quantify the automatability of candidate routines based on data attributes.

Acknowledgments. Work supported by the European Research Council (PIX project) and by the Australian Research Council (DP180102839).

References

[1] H. Leopold, H. van der Aa, and H. A. Reijers, "Identifying candidate tasks for robotic process automation in textual process descriptions," in *BPMDS*. Springer, 2018, pp. 67–81.

[2] V. Leno, A. Polyvyanyy, M. Dumas, M. La Rosa, and F. M. Maggi, "Robotic process mining: Vision and challenges," *Business & Information Systems Engineering*, 2020.

[3] H. Dev and Z. Liu, "Identifying frequent user tasks from application logs," in *IUI*. Springer, 2017, pp. 263–273.

[4] A. Jimenez-Ramirez, H. A. Reijers, I. Barba, and C. Del Valle, "A method to improve the early stages of the robotic process automation lifecycle," in *CAiSE*. Springer, 2019, pp. 446–461.

[5] A. Bosco, A. Augusto, M. Dumas, M. La Rosa, and G. Fortino, "Discovering automatable routines from user interaction logs," in *BPM Forum*. Springer, 2019.

[6] J. Gao, S. J. van Zelst, X. Lu, and W. M. van der Aalst, "Automated robotic process automation: A self-learning approach," in *OTM Confederated International Conferences*. Springer, 2019, pp. 95–112.

[7] V. Leno, A. Polyvyanyy, M. La Rosa, M. Dumas, and F. M. Maggi, "Action logger: Enabling process mining for robotic process automation," in *BPM Demos*. CEUR, 2019.

[8] C. Linn, P. Zimmermann, and D. Werth, "Desktop activity mining-a new level of detail in mining business processes," in *Workshops der INFORMATIK 2018-Architekturen, Prozesse, Sicherheit und Nachhaltigkeit*. Köllen Druck+ Verlag GmbH, 2018.

[9] H. Cao, N. Mamoulis, and D. W. Cheung, "Discovery of periodic patterns in spatiotemporal sequences," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 4, pp. 453–467, 2007.

[10] Y. Zhu, M. Imamura, D. Nikovski, and E. Keogh, "Matrix profile vii: Time series chains: A new primitive for time series data mining," in *ICDM*. IEEE, 2017, pp. 695–704.

[11] M. Spiliopoulou, B. Mobasher, B. Berendt, and M. Nakagawa, "A framework for the evaluation of session reconstruction heuristics in web-usage analysis," *Informs journal on computing*, vol. 15, no. 2, pp. 171–190, 2003.

[12] D. Bayomie, A. Awad, and E. Ezat, "Correlating unlabeled events from cyclic business processes execution," in *CAiSE*. Springer, 2016, pp. 274–289.

[13] D. R. Ferreira and D. Gillblad, "Discovering process models from unlabelled event logs," in *BPM*. Springer, 2009, pp. 143–158.

[14] J. Han, H. Cheng, D. Xin, and X. Yan, "Frequent pattern mining: current status and future directions," *Data mining and knowledge discovery*, vol. 15, no. 1, pp. 55–86, 2007.

[15] S. D. Lee and L. De Raedt, "An efficient algorithm for mining string databases under constraints," in *International Workshop on Knowledge Discovery in Inductive Databases*. Springer, 2004, pp. 108–129.

[16] E. Ohlebusch and T. Beller, "Alphabet-independent algorithms for finding context-sensitive repeats in linear time," *Journal of Discrete Algorithms*, vol. 34, pp. 23–36, 2015.

[17] J. Wang and J. Han, "Bide: Efficient mining of frequent closed sequences," in *ICDE*. IEEE, 2004, pp. 79–90.

[18] F. Fumarola, P. F. Lanotte, M. Ceci, and D. Malerba, "Clofast: closed sequential pattern mining using sparse and vertical id-lists," *Knowledge and Information Systems*, vol. 48, no. 2, pp. 429–463, 2016.

[19] V. Leno, M. Dumas, M. La Rosa, F. M. Maggi, and A. Polyvyanyy, "Automated discovery of data transformations for robotic process automation," *ArXiv*, vol. abs/2001.01007, 2020.

[20] M. Sharir, "A strong-connectivity algorithm and its applications in data flow analysis," *Computers & Mathematics with Applications*, vol. 7, no. 1, pp. 67–72, 1981.

[21] B. Dalmas, N. Tax, and S. Norre, "Heuristic mining approaches for high-utility local process models," *T. Petri Nets and Other Models of Concurrency*, vol. 13, pp. 27–51, 2018.

Anomaly Detection on Event Logs with a Scarcity of Labels

Sylvio Barbon Junior*, Paolo Ceravolo†, Ernesto Damiani‡, Nicolas Jashchenko Omori* and Gabriel Marques Tavares†
*Londrina State University (UEL), Londrina, Brazil
Email: {barbon, omori}@uel.br
†Università degli Studi di Milano (UNIMI), Milan, Italy
Email: {paolo.ceravolo, gabriel.tavares}@unimi.it
‡Khalifa University (KUST), Abu Dhabi, UAE
Email: ernesto.damiani@kustar.ac.ae

***Abstract*—Assuring anomaly-free business process executions is a key challenge for many organizations. Traditional techniques address this challenge using prior knowledge about anomalous cases that is seldom available in real-life. In this work, we propose the usage of *word2vec* encoding and *One-Class Classification* algorithms to detect anomalies by relying on normal behavior only. We investigated 6 different types of anomalies over 38 real and synthetics event logs, comparing the predictive performance of Support Vector Machine, One-Class Support Vector Machine, and Local Outlier Factor. Results show that our technique is viable for real-life scenarios, overcoming traditional machine learning for a wide variety of settings where only the normal behavior can be labeled.**

***Keywords*-One Class Classification; anomaly detection; Local Outlier Factor; encoding; Support Vector Machine**

I. Introduction

In Business Process Management (BPM), anomaly detection has traditionally focused on deviations from a reference model [1]. The reference to a business process model is significant as organizations are driven by normative conformity and adherence to plans. In addition, a process model is an abstraction that can specify sequential, conditional, iterative, and concurrent behavior. This way, cases that significantly differ in the sequence or number of activities can still comply with the same model [2]. Thus, model-aware analytics, such as Process Mining (PM) [3], generalize better, in the BPM domain, than traditional statistics or data mining techniques.

The input of PM is an event log, composed of recorded events described by the *activity* executed at a certain time, the actors who executed the activity, the associated resources, and costs. A unique sequence of time-ordered events of a same case is called *trace*. The analysis of the conformance between a trace and a process model, known in PM as *conformance checking*, highlights control-flow and data-flow anomalies. Control-flow anomaly refers to activities executed in the wrong order. Data-flow anomaly refers to unexpected values in events' attributes, for instance, a user who executes an activity he does not have access to. Patterns in the event log may indicate the root cause of an anomaly, such as a system malfunctioning or a security violation [4].

However, a known problem of PM is that in real scenarios the process model may not be available, be inadequate or incorrect, or significant domain knowledge may be required to elicit it [5]. More recently, Machine Learning (ML) has been proposed as an alternative to process-aware methods, as it can learn anomalies directly from the event log, without needing a reference model [6]. The training stage of ML requires, however, to hold enough ground truth labels for the output classes to be learned. Hence, the need for significant domain knowledge comes back as a need for pre-labeled data.

To get rid of this issue, in this paper, we propose to combine vector space models and One-Class Classification (OCC) algorithms to run anomaly detection by relying solely on the normal behavior described in an event log. Inspired by natural language processing (NLP) we treat activities as words and traces as sentences and use the word2vec algorithm to map them into a vector space [7]. When traces are encoded in the vector space, we use OCC methods to detect anomalies.

To evaluate our approach, we compared OCC to supervised ML. In our experiments, we used both synthetic and real-world event logs, including six different types of anomalies. Results show that our semi-supervised approach handles anomaly detection with performances similar to supervised ML, or even better for a wide variety of anomalies.

The paper is organized as follows. Section II presents the current state of the art. Section III presents our methodology and the datasets used in the experiments. Section IV reports our results and evaluates them with a detailed analysis of the advantages of combining vector space encoding and OCC methods. Section V evaluates the overall implications of the results and places the method in the literature. Finally, we draw our conclusions in Section VI.

II. Related Works

Much BPM research relies on control-flow or data-flow verification for anomaly detection, using place/transition nets or constraint satisfaction [8], [9], [10]. Although various terms

This study was financed in part by Coordination for the National Council for Scientific and Technological Development (CNPq) of Brazil - Grant of Project 420562/2018-4 and Fundação Araucária (Paraná, Brazil). It was also partly supported by the program "Piano di sostegno alla ricerca 2019" funded by Universita degli Studi di Milano.

978-1-7281-9832-3/20/$31.00 ©2020 IEEE
DOI 10.1109/ICPM49681.2020.00032

are used in the literature to frame these issues, PM has been largely adopted in the last years [1].

Bezerra et al. [11] define anomaly in PM using a set of assumptions: (i) the set of traces can be partitioned into a set of normal and a set of anomalous executions; (ii) each of the anomalous execution is infrequent compared to the set of all executions; (iii) the process model representing normal behavior should "make more sense" than the process model representing anomalous traces. More recently, an analysis of online anomaly detection in PM has also been proposed [12]. One of the earliest works for anomaly detection in PM uses a conformance checking pipeline [13], [11]. First, the method filters the dataset based on domain-dependent knowledge. From that, process discovery techniques are applied to the filtered log. Then, the most appropriate model (high fitness while structurally simple) is chosen as the process model. Finally, traces are classified depending on model fitting; that is, a non-fitting trace is classified as anomalous, whereas a fitting trace is classified as a normal execution. However, this approach depends on a clean event log for model creation and assumes that process discovery techniques might generate an ideal model, which is not necessarily true. Moreover, domain knowledge costs resources and is not always available, hence making this method impractical in most situations.

Likelihood graphs encoding the control-flow of a process can be exploited to identify traces with activities having direct follow relationships deviating from an observed probability [14]. However, parallel and recursive behavior cannot be correctly handled using these techniques. For this reason, many PM researchers adopt model-aware analytics where traces are replayed over the model. Each executed activity corresponds to consuming a token. Consumed, missed, and remaining tokens are counted to calculate conformance statistics [3]. The accuracy of this approach substantially depends on the quality of the process model but model discovery techniques are the result of a trade-off between precision and generality. Relying on automatically generated models for anomaly detection is then incongruous while cleansing these models by manual effort is costly.

The adoption of methods using ML for detecting anomalies in business processes has also risen in recent years. In [6], the authors use an autoencoder (a class of neural networks) to model process behavior and further detect anomalies. More recently, a recurrent neural network architecture was proposed [15], [16]. The method (BINet) handles both control-flow and the data perspective of a business process, detecting anomalies in both situations. In its core, BINet is a Gated Recurrent Unit trained to predict the next event. In the preprocessing stage, the logs are transformed into a numerical representation using integer encoding. Anomalies are detected by assuming that an anomalous attribute has a lower probability than a normal attribute, and regulate it by a heuristic threshold. The experiments show that BINet usually outperforms other methods. However, there is a very high computational cost given the deep learning architecture. Moreover, the authors note that BINet suffers from a forgetting problem when sequences of activities are repeated in a case.

A great part of ML techniques relies on supervised strategies, depending on labels and specialist intervention as a preliminary step. In real life, the true class label of a case is not immediately available (or will never be available), and manually labeling might be labor-intensive. Thus, anomaly detection on event logs has to cope with label scarcity. A potential alternative to supervised ML is working with unsupervised learning, particularly clustering methods. Cluster labels have been used as automatically computed labels. Following this idea, in [17], authors developed a framework aimed at bridging the gap between the abstraction levels of row data logs and process models. However, the results of clustering solutions have a high level of variability between different randomized executions and parameters [18].

One-Class Classification algorithms emerged as a suitable solution to traditional supervised learning, as well as being able to address the main drawbacks of the clustering algorithm. OCC is computationally more competitive because only the common behavior samples are required in the training stage [19]. In the PM scenario, these can be achieved from the main business process model and its variants automatically generated. OCC can be performed by either a density or a boundary strategy. When defining the classification boundary around the positive class towards accepting as many samples as possible, it minimizes the chance of accepting non-positive (i.e. outlier) samples [19], [20]. Thus, OCC is a suitable and straightforward solution to address scenarios with a scarcity of labels.

Our approach overcomes the mentioned gaps with: (i) usage of a better encoding technique; (ii) ability to work with a scarcity of labels; (iii) no need of a process model; (iv) no required expert knowledge about the business process.

III. Methodology

This section presents the event logs, the encoding strategy, the algorithms, and the evaluation metrics we used for comparing OCC to supervised ML.

A. Event logs

A controlled scenario with synthetic event logs and known labels is ideal to evaluate the performance of our method.

We generated event logs following the procedure proposed by Nolle et al. [16] that allows creating events logs from process models using a seed state, where the same seed guarantees the creation of identical event logs. A likelihood graph [14] models the dependencies within events and between events and attributes. This implies that the probability distributions in the control-flow are constrained. For example, activity A has a 60% probability of being followed by activity B. The constraints also bind events and attributes, e.g., an activity A has an 80% probability of being followed by activity B when A is executed on Mondays. This modeling allows introducing long-term control-flow dependencies, typical of real-world scenarios. Event logs can be generated by merely performing random walks in the likelihood graph, respecting

its transition probabilities. We decided to replicate previously proposed datasets to create a common ground for comparison between different methods. The steps followed to replicate this synthetic dataset are accessible in the code repository linked to [16]. The models we employed are a subset of those used in [16] but cover the entire range of those publicly available. With the use of the PLG2 tool [21], six random process models were created. They vary in complexity, i.e., number of activities, breadth, and width. A handmade procurement process model (P2P) from [15] was also added. In addition to these synthetic event logs, we used real-life event logs from previous Business Process Intelligence Challenges (BPIC)[1]: BPIC12, BPIC13, BPIC15, and BPIC17.

After the event logs were produced, the next step was to inject anomalies into them. Following [11], [14], [16], we applied six different anomaly types. (i) *Early*: a sequence of 2 or fewer events executed too early, which is then skipped later in the case. (ii) *Late*: a sequence of 2 or fewer events executed too late. (iii) *Insert*: 3 or less random activities inserted in the case. (iv) *Skip*: a sequence of 3 or less necessary events is skipped. (v) *Rework*: a sequence of 3 or less necessary events is executed twice. (vi) *Attribute*: an incorrect attribute value is set in 3 or fewer events.

These anomalies were injected in 30% of all cases from each event log, including those from real-life scenarios (BPICs). Note that the anomalies are on the event level, but they can be converted to the case level easily: a case is anomalous if any of its event attributes is anomalous. For each synthetic process model, four likelihood graphs were created with different transition probabilities, dependencies, and the number of attributes. In total, we created 28 synthetic event logs that, together with the BPICs, form a total of 38 event logs used in our experimental evaluation. Table I shows the detailed statistics of these event logs.

Name	#Logs	#Activities	#Cases	#Events	#Attrs.	#Attr. values
P2P	4	27	5k	48k-53k	1-4	13-386
Small	4	41	5k	53k-57k	1-4	13-360
Medium	4	65	5k	39k-42k	1-4	13-398
Large	4	85	5k	61k-68k	1-4	13-398
Huge	4	109	5k	47k-53k	1-4	13-420
Gigantic	4	154-157	5k	38k-42k	1-4	13-409
Wide	4	68-69	5k	39k-42k	1-4	13-382
BPIC12	1	73	13k	290k	0	0
BPIC13	3	11-27	0.8k-7.5k	4k-81k	2-4	23-1.8k
BPIC15	5	422-486	0.8k-1.4k	46k-62k	2-3	23-481
BPIC17	1	53	31k	1.2M	1	299

TABLE I: Detailed event logs statistics

B. Encoding strategy

Traditionally, ML and data mining techniques cannot be directly applied to PM event logs due to a mismatch at the representation level [22]. ML operates at the event level, where each event is an instance, PM at the business case level, where a group of events represents an instance of the process. This way, embedding techniques are necessary to overcome this gap [23].

There are a few attempts to perform event log encoding in the literature. In [6], the authors use one-hot encoding to transform the log into a numerical representation before feeding an autoencoder. Similarly, in [16], [15], the authors use integer encoding to map all log attributes into integers and then use the resulting vectors as the input for a deep learning algorithm. Though both encoding approaches are traditional in ML literature, there are better-performing ones proposed recently. In [24], the authors represent a trace using the *doc2vec* approach [25], which extends the Continuous Bag of Words architecture with a paragraph vector for encoding traces. Inspired by this perspective, we employed the widely known word2vec algorithm [7]. Word2vec encodes words by training a neural network to reconstruct its linguistic context in a corpus. Word vectors embed the weights learned by the neural network capturing, this way, semantic and syntactic similarities in vectors. Our method interprets each activity as a word. Hence, the corpus is composed of the set of unique activities in the business process. The vectors we obtain can then be aggregated to get trace-level representations. In our case, we use the element-wise mean, i.e., the trace representation is the mean of its activities weights.

C. Classification algorithms

We used three different algorithms for classifying the traces into normal and anomalous, Support Vector Machines (SVM) [26], One-Class Support Vector Machines (OCSVM) proposed by Tax et al. [27] and Local Outlier Factor (LOF) proposed by Breunig et al. [28]. All three algorithms are available on the Scikit-learn Python package [29]. Several hyperparameter values were combined as a tuning strategy, as illustrated in Table II. The first technique is supervised, meaning that it depends on labels for its training. The other two algorithms require only the common behavior to induce a model able to classify unlabeled data. The SVM is used as a baseline for comparison purposes, aiming to check if the techniques that do not depend on completely labeled data can produce comparable results.

TABLE II: Collection of combined hyperparameter values

Method	Hyperparameter	Values
SVM	*c*	[0.1, 1, 10, 100, 1000, 10000, 100000]
	kernel	[polynomial, rbf, sigmoid]
	gamma	[auto, scale]
OCSVM	*nu*	[0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4]
	kernel	[polynomial, rbf, sigmoid]
	gamma	[auto, scale]
LOF	*k*	[1, 10, 25, 50, 100, 250]
	contamination	[0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, auto]

SVM considers each n-dimensional input vector as a point in an n-dimensional space. Then, the algorithm searches for an n-1 dimensional hyperplane that can correctly separate points. To find this hyperplane, the algorithm tries to maximize the

[1]https://www.tf-pm.org/resources/logs

distance of the hyperplane to the closest boundary points, called Support Vectors. In this work we used the sigmoid, polynomial and RBF kernels for the tests, varying the c and $gamma$ hyperparameters.

OCSVM is a one-class version of SVM. Similarly, it considers the input vectors as points in an n-dimensional space but, instead of creating a hyperplane to separate the data into two classes, it creates an n-dimensional hypersphere that can represent the input data such as the points that are inside the hypersphere are considered the normal behavior and the points that are outside are considered anomalies. The algorithm aims at maximizing the number of points inside the hypersphere while minimizing the empty space inside it. Different values for the parameter nu are used to assume different levels of generalization by the algorithm, with higher values standing for higher generalization. The sigmoid, polynomial and RBF kernels were considered in our experiments.

LOF detects outliers given a set of vectors, which are also treated as points in an n-dimensional space. The main idea behind LOF is to compare the local point density to its k nearest neighbors densities. Thus, points with lower density than its neighbors are considered outliers. The experiment used different numbers of k nearest neighbors as well as different values for *contamination*, a hyperparameter that defines the density threshold for anomaly detection. For the evaluation of the algorithms, we chose the F-score metric [30].

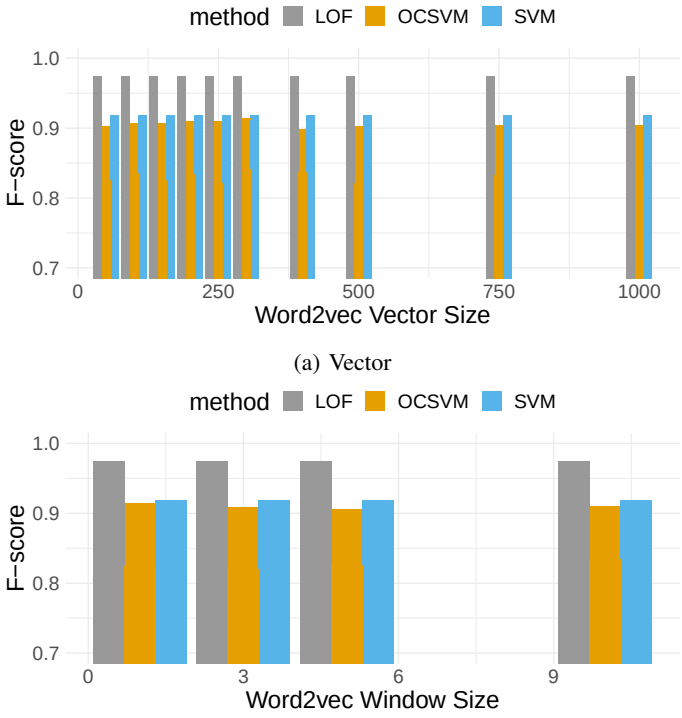
IV. Results

This section presents the results obtained from several complementary perspectives, evaluating their impact on the overall performance. To follow the open science principles, we made the experiments and event logs available[2].

A. Word2vec descriptive performance

The initial experiment aims at evaluating how the proposed encoding influences the results. Regarding word2vec hyperparameters, we explored the number of dimensions and window size, which are reportedly the most impacting hyperparameters. The implementation used the *gensim* package for Python[3] and the other hyperparameters were set to standard values. Figure 1a and 1b show the F-score values obtained.

According to Figure 1a, the impact of different vector sizes on the performance is minimal. For this task, we ranged values from 50 to 1000. In other domains, the number of dimensions highly affects the representational capacity of word2vec. A low number usually diminish encoding quality. However, traditional NLP tasks perform over text corpora containing large quantities of unique words. In business processes, the set of unique activities (which we consider as words) is of several orders of magnitude less. Moreover, the contexts that surround business process activities are often less heterogeneous. The same phenomenon is revealed in Figure 1b, where various window sizes are compared. The window size controls how much of the context surrounding each word is considered in the learning procedure. We expected longer windows to drive richer characterization of activities, but our experimental results show that it does not impact performances.



(a) Vector

(b) Window

Fig. 1: F-score obtained with different window and vector sizes of word2vec across all event logs

The main lesson learned from these experiments is the robustness of word2vec in representing the context of business activities. It can be adopted without the need to search for optimal hyperparametrization. Hence, in BPM applications, smaller vector and window sizes are advised as they consume less computational resources without losing representational capacity.

Figure 2 further demonstrates the effectiveness of word2vec embeddings. We applied this technique to one of the `small` event log using 200 dimensions and a window size of 1. Then, using the t-SNE dimensionality reduction technique (with standard hyperparameters from Scikit-learn package[4]), the number of dimensions was reduced to two. We can see how word2vec distributes the normal and anomalous classes in the feature space. Word2vec correctly interprets the activities contexts placing anomalous behavior apart from normal. Moreover, anomalous instances are usually near normal ones because they are a slight variation of them, according to the injecting procedure we followed.

[2]https://github.com/gbrltv/ProcessAnomalyDetector
[3]https://radimrehurek.com/gensim/models/word2vec.html
[4]https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

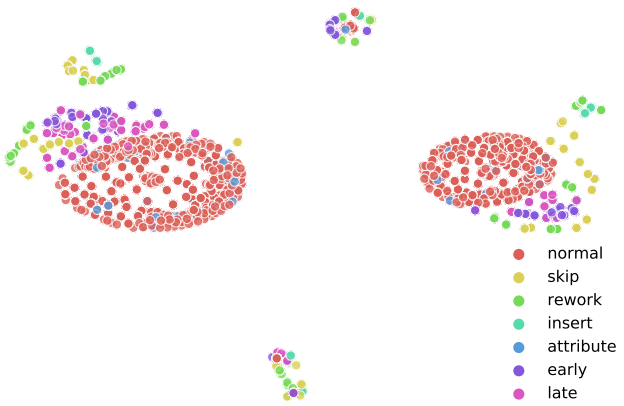
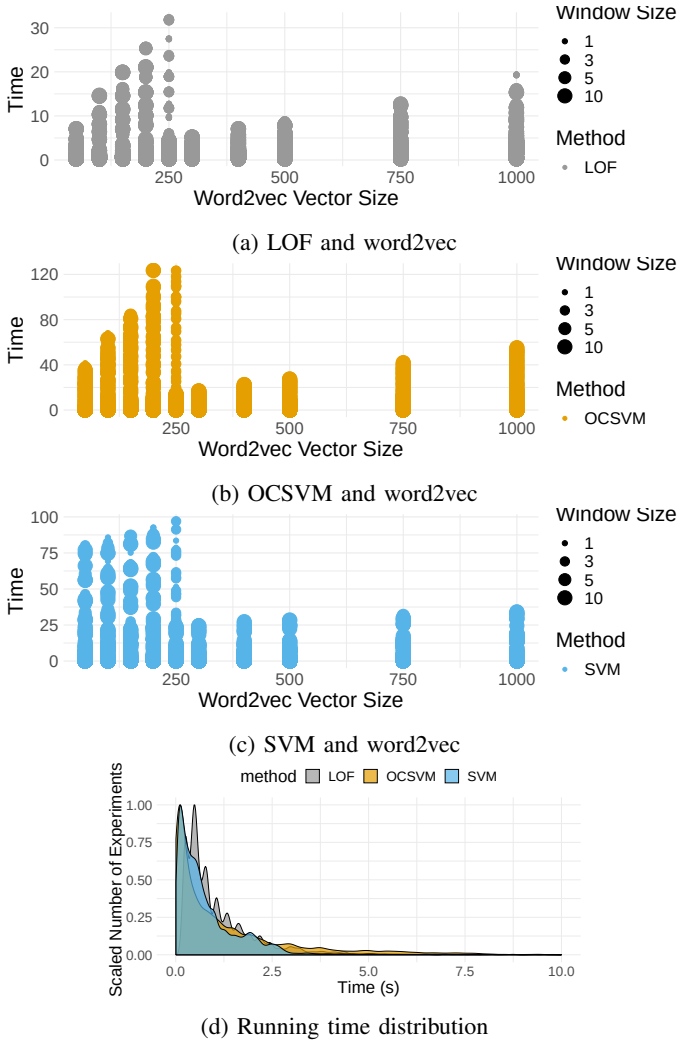


Fig. 2: `small` trace distribution. The experiment used word2vec to model the business process behavior. Then, the t-SNE dimensionality reduction technique was applied for visualization. It is notable how anomalous and normal behavior is quite separated in the feature space



(a) LOF and word2vec

(b) OCSVM and word2vec

(c) SVM and word2vec

(d) Running time distribution

Fig. 3: Comparing execution time, in seconds, with window and vector size per algorithm across all event logs. Quadrant (d) reports the likelihood of observing a specific execution time in the experiments we ran

B. Time analysis

We compared a total of 40 different word2vec configurations with several setups of LOF, OCSVM, and SVM. It is possible to observe that small vector sizes, i.e., less than 200 features, affect the execution time with small window sizes. Configurations of this kind demand more time due to more frequent sliding over the samples, as seen in figures 3a, 3b and 3c. When dealing with more than 250 features, time consumption is independent of window sizes, drastically reducing the number of outliers in the results.

SVM shows a slightly superior time performance, followed by LOF and OCSVM. However, the average time difference is less than one second, as observed in the running time distribution of Figure 3d. The size of the vector has, indeed, the most noticeable impact on execution time. We also compared the time of classification algorithms according to the Friedman and Nemenyi test [31]. Using $\tau = 0.05$ and critical distance of 0.53, it was possible to account a significantly different performance of SVM (1.37, ranked first), the fastest algorithm. LOF (2.13, ranked second) and OCSVM (2.50, ranked third) were not accounted as significantly different.

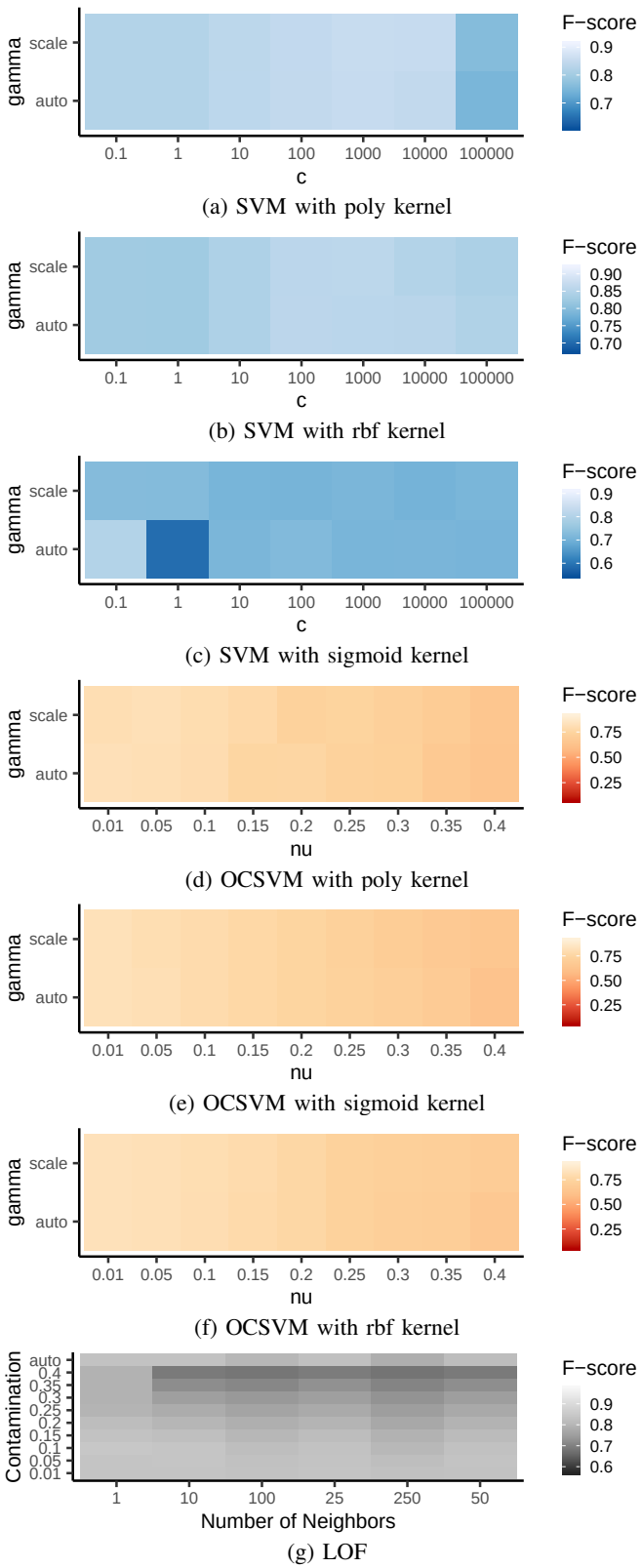
Considering the results of Section IV-A, we then suggest the adoption of small vector sizes, particularly 50 features, which provide fast processing and stability for most event logs with competitive predictive performance.

C. Hyperparameter selection

This experiment focuses on the tuning of hyperparameters to improve performances and support fair comparison among the methods. Regarding SVM, the best hyperparameter to deal with all event logs was c =1000 and $gamma$ as $scale$ using a $polynomial$ kernel. This algorithm was the most volatile to hyperparameter selection, e.g., some configurations of c using $sigmoid$ kernel reduced drastically the F-score, as shown in Figures 4a, 4b and 4c. For OCSVM, the nu hyperparameter has impacted the most on performance. Small nu values (best $nu = 0.01$) results in better predictive outcomes independent of $gamma$ or $kernel$. As illustrated in Figures 4d, 4e and 4f. Finally LOF, similarly to SVM, demands a combination of hyperparameters to achieve the optimal performance. Also, the $auto$ value for the $contamination$ parameter obtained an average good performance. For k (number of neighbors), smaller values (1, 10, 25 and 50) were the best. Using high $contamination$ (value of 0.4) we obtained the worst LOF performances, as visible in Figure 4g.

D. Classification performance

Figure 5 shows a boxplot of the best F-scores for each algorithm. LOF's entire first quartile is above 0.95 while its median is 0.96. The median F-scores for OCSVM and SVM were 0.87 and 0.9, respectively. Therefore, LOF outperforms



(a) SVM with poly kernel

(b) SVM with rbf kernel

(c) SVM with sigmoid kernel

(d) OCSVM with poly kernel

(e) OCSVM with sigmoid kernel

(f) OCSVM with rbf kernel

(g) LOF

Fig. 4: Hyperparameter overview of SVM, OCSVM, and LOF, light regions means best predictive performance (F-score) across all event logs

both SVM and OCSVM. This result is valuable as LOF does not even need anomalous examples to induce its model, making it easier to prepare an event log for this method. That is, supervised methods do not necessarily imply better performance.

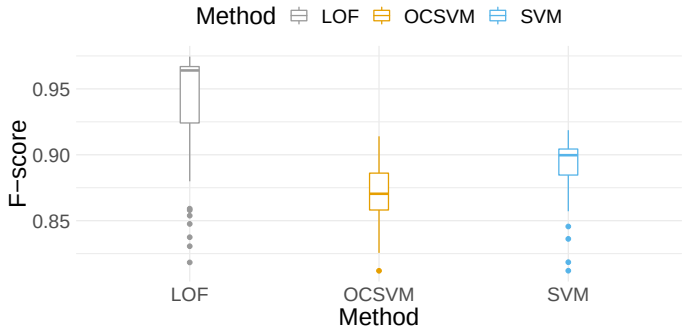


Fig. 5: Boxplot F-score results for each classification algorithm across all event logs

Figure 6 compares the performances over individual event logs. As corroborated by the previous analysis, LOF outperforms the other two algorithms in almost all event logs. This trend is even more explicit in synthetic event logs, where LOF reaches F-scores of over 0.95. In most datasets, SVM is better than OCSVM by a low margin, however, it required a longer tuning process for its hyperparameters. Real-life event logs have a lower F-score when compared to synthetics. The main reason is that BPIC event logs naturally contain not labeled anomalies, which incorrectly represent normal behavior.

F-score comparison among the classification algorithms, according to the Friedman and Nemenyi test using $\tau = 0.05$, showed significantly different performances. The critical distance of 0.53 attested the superiority of LOF (1.11, ranked first), followed by SVM (2.00, ranked second) and OCSVM (2.89, ranked third).

E. Performances by anomaly

As there are six anomaly types, we further experimented to analyze their differences. For this, we created six additional event logs using the `medium` behavior. For each anomaly type, we created an event log with 30% of anomalies using only the respective type. Figure 7 reports the results of the algorithms for each anomaly. The first important note is that having only one type of anomaly in the log makes the anomaly identification easier.

The *attribute* anomalies are the most difficult to detect as they do not insist on the control-flow perspective. In spite of it, the results obtained are not particularly severe. Regarding the other anomaly types, LOF can detect anomalous instances as they all affect the control-flow aspect, which is well inferred by word2vec. SVM presented good results for *insert*, *rework* and *skip* anomalies. These anomalies are easier to detect due to a higher impact on activities' contexts. For instance, *skip* makes a trace to miss a required activity execution. Then, when analyzing a trace with a skipped activity, word2vec modeling

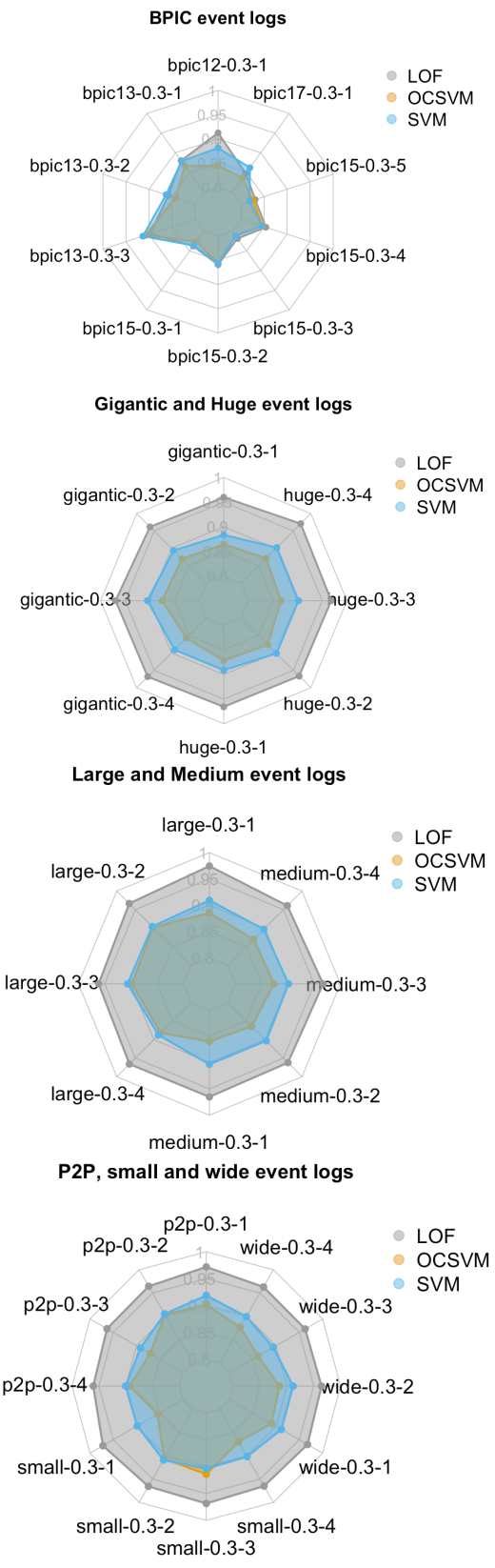


Fig. 6: F-score of all event logs grouped by size and behavior

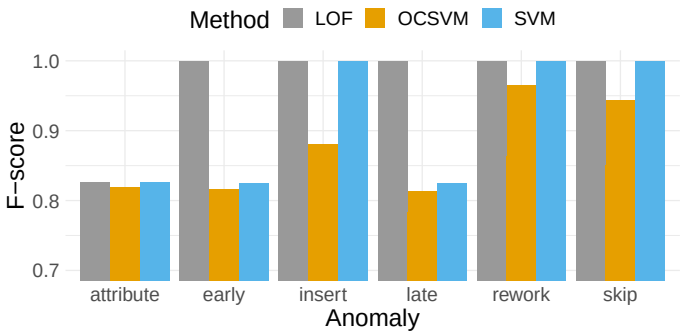


Fig. 7: Comparing the F-score of each algorithm using the event log `medium`

is sensible enough to detect that the trace context is different from normal behavior. For *early* and *late* anomalies, the effect on the context is more subtle because the activities are still in the trace even if in wrong positions. Therefore, this shows the importance of having an OCC classification on top of word2vec as it relies on normal samples, counterbalancing this issue. This is seen by the exceptional performance reached by LOF.

V. Discussion

Overall, SVM was slightly faster than OCC methods, however, SVM performance is significantly dependent on hyperparameters, e.g., good performance was achieved at the cost of a long tuning process. When comparing accuracy, LOF was superior, reaching the best predictive performance for most real-life and all the synthetic event logs. The SVM performance was competitive only in a few real-life datasets, though the pre-existence of non-reported anomalous cases can have swayed the learning process from OCC methods. OCSVM was the slowest and less predictive approach. It is important to mention that *early* and *late* anomalies, when compared in a synthetic log of medium size, related to a similar detection score. These anomalies are more difficult to be modeled since the events affected are still present in the trace with changed positions. For this reason, when aggregating the trace representation, the overall trace encoding can be similar to a normal trace. The phenomenon is observable because word2vec is more flexible to ordering than traditional conformance checking algorithms. That is, word2vec context composes the activities neighborhood more broadly, allowing for different sequences to have similar representations in some cases. However, when combining word2vec with LOF, this challenge was overcome, as Figure 7 shows. Another similar set was *insert*, *rework* and *skip*, that obtained higher detection scores. These anomalies are easily captured by word2vec because they insert an abnormal activity, repeat the execution of an activity, and skip an expected execution, respectively. Thus, the difference between normal behavior is more abrupt. The worst scores were observed with the *attribute* anomaly type. The *attribute* anomaly does not affect the control-flow perspective, it only affects the data attributes. This way, it is

more complex for encoders processing traces to capture this anomaly.

Deep learning methods [6], [16], [15] are considered out of the scope of this paper and we did not compare to the algorithms studied in this work. The reason is deep learning algorithms require ground truth labels and large availability of computational resources. They, in other words, match to application requirements completely different from those we assumed.

VI. Conclusion

The presented work proposes the use of word2vec, a traditional NLP technique, to encode business process behavior as the context of activities in an event log. Moreover, the method explores the use of OCC algorithms, showing their advantage in scenarios with a scarcity of labels. The performance obtained by the combination of trace encoding and OCC techniques demonstrates the feasibility of our method. Therefore, organizations can profit by the use of this methodology as detecting anomalies is a significant concern given their impact on process performance and resource consumption. Different word2vec configurations were quite similar in terms of predictive performance, demonstrating robustness to vector and window sizes. When comparing to supervised ML, OCC presented a better performance in most cases. As future work, we plan (i) to explore other aggregation techniques to represent a trace encoding, (ii) consider the data-flow perspective, and (iii) to apply OCC algorithms in online PM.

References

[1] W. M. P. van der Aalst, *Process Mining: Data Science in Action*, 2nd ed. Heidelberg: Springer, 2016.

[2] M. Boltenhagen, T. Chatain, and J. Carmona, "Generalized alignment-based trace clustering of process behavior," in *International Conference on Applications and Theory of Petri Nets and Concurrency*. Springer, 2019, pp. 237–257.

[3] F. Bezerra, J. Wainer, and W. M. van der Aalst, "Anomaly detection using process mining," in *Enterprise, business-process and information systems modeling*. Springer, 2009, pp. 149–161.

[4] R. P. Jagadeesh Chandra Bose and W. van der Aalst, "Trace alignment in process mining: Opportunities for process diagnostics," in *Business Process Management*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 227–242.

[5] W. M. Van der Aalst, V. Rubin, H. Verbeek, B. F. van Dongen, E. Kindler, and C. W. Günther, "Process mining: a two-step approach to balance between underfitting and overfitting," *Software & Systems Modeling*, vol. 9, no. 1, p. 87, 2010.

[6] T. Nolle, S. Luettgen, A. Seeliger, and M. Mühlhäuser, "Analyzing business process anomalies using autoencoders," *Machine Learning*, vol. 107, no. 11, pp. 1875–1893, 2018.

[7] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *CoRR*, vol. abs/1301.3781, 2013.

[8] S. Barbon Junior, G. M. Tavares, V. G. T. da Costa, P. Ceravolo, and E. Damiani, "A framework for human-in-the-loop monitoring of concept-drift detection in event log stream," in *Companion Proceedings of the The Web Conference 2018*, ser. WWW '18. International World Wide Web Conferences Steering Committee, 2018, p. 319–326.

[9] A. Rogge-Solti and G. Kasneci, "Temporal anomaly detection in business processes," in *Business Process Management*. Cham: Springer International Publishing, 2014, pp. 234–249.

[10] M. Rovani, F. M. Maggi, M. De Leoni, and W. M. Van Der Aalst, "Declarative process mining in healthcare," *Expert Systems with Applications*, vol. 42, no. 23, pp. 9236–9251, 2015.

[11] F. Bezerra and J. Wainer, "Algorithms for anomaly detection of traces in logs of process aware information systems," *Information Systems*, vol. 38, no. 1, pp. 33 – 44, 2013.

[12] P. Ceravolo, G. Marques Tavares, S. B. Junior, and E. Damiani, "Evaluation goals for online process mining: a concept drift perspective," *IEEE Transactions on Services Computing*, pp. 1–1, 2020.

[13] F. Bezerra, J. Wainer, and W. M. P. van der Aalst, "Anomaly detection using process mining," in *Enterprise, Business-Process and Information Systems Modeling*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 149–161.

[14] K. Böhmer and S. Rinderle-Ma, "Multi-perspective anomaly detection in business process execution events," in *On the Move to Meaningful Internet Systems: OTM 2016 Conferences*. Cham: Springer International Publishing, 2016, pp. 80–98.

[15] T. Nolle, A. Seeliger, and M. Mühlhäuser, "Binet: Multivariate business process anomaly detection using deep learning," in *Business Process Management*. Cham: Springer International Publishing, 2018, pp. 271–287.

[16] T. Nolle, S. Luettgen, A. Seeliger, and M. Mühlhäuser, "Binet: Multi-perspective business process anomaly classification," *Information Systems*, p. 101458, 2019.

[17] G. Tello, G. Gianini, R. Mizouni, and E. Damiani, "Machine learning-based framework for log-lifting in business process mining applications," in *Business Process Management*. Cham: Springer International Publishing, 2019, pp. 232–249.

[18] C. C. Aggarwal, "Proximity-based outlier detection," in *Outlier Analysis*. Springer, 2017, pp. 111–147.

[19] S. S. Khan and M. G. Madden, "A survey of recent trends in one class classification," in *Irish conference on artificial intelligence and cognitive science*. Springer, 2009, pp. 188–197.

[20] I. Kang, M. K. Jeong, and D. Kong, "A differentiated one-class classification method with applications to intrusion detection," *Expert Systems with Applications*, vol. 39, no. 4, pp. 3899–3905, 2012.

[21] A. Burattin, "Plg2: Multiperspective processes randomization and simulation for online and offline settings," 2015.

[22] G. M. Tavares, P. Ceravolo, V. G. Turrisi Da Costa, E. Damiani, and S. Barbon Junior, "Overlapping analytic stages in online process mining," in *2019 IEEE International Conference on Services Computing (SCC)*, July 2019, pp. 167–175.

[23] P. Ceravolo, E. Damiani, M. Torabi, and S. Barbon, "Toward a new generation of log pre-processing methods for process mining," in *Business Process Management Forum*. Cham: Springer International Publishing, 2017, pp. 55–70.

[24] P. De Koninck, S. vanden Broucke, and J. De Weerdt, "act2vec, trace2vec, log2vec, and model2vec: Representation learning for business processes," in *Business Process Management*. Cham: Springer International Publishing, 2018, pp. 305–321.

[25] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ser. ICML'14. JMLR.org, 2014, p. II–1188–II–1196.

[26] C. Cortes and V. Vapnik, "Support vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.

[27] D. M. Tax and R. P. Duin, "Support vector data description," *Machine Learning*, vol. 54, no. 1, pp. 45–66, Jan 2004.

[28] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: Identifying density-based local outliers," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '00. New York, NY, USA: Association for Computing Machinery, 2000, p. 93–104.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[30] A. Tharwat, "Classification assessment methods," *Applied Computing and Informatics*, 2018.

[31] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *The Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.

TOAD: Trace Ordering for Anomaly Detection

Florian Richter, Yifeng Lu, Ludwig Zellner, Janina Sontheim, and Thomas Seidl
Ludwig-Maximilians-Universität München, Munich, Germany
Email: {richter, lu, zellner, sontheim, seidl}@dbs.ifi.lmu.de

***Abstract*—Outlier detection is one of the most important tasks to keep your processes in control. Unawareness of critical anomalies can lead to exhausting expenses, hence, it is highly beneficial to treat process failures as soon as possible. However, anomalies are difficult to detect due to their rarity though they occur too often to neglect the necessity of its detection. Even if the detection problem is solved, the treatment of singular anomalies and the adjustment of the process based on each abnormal trace is tedious and costly regarding time and money. To increase the efficiency of later anomaly treatment, we propose a novel strategy to detect collective anomalies. However, this is not equivalent to anomaly clustering as a post-processing step. TOAD orders process instances by similarity and detects abnormal accumulations of deviating cases. These collections are abnormal due to their aggregated behavior. Assuming that similar deviations are caused by the same reason, the treatment of such an anomaly is more cost-efficient than the handling of deviating singletons. Applying TOAD to an event log yields a ranking of significant, temporally abnormal trace collections, that provide a baseline for further analysis.**

1. Introduction

Deviations exist in probably every ever so well-defined process. On the one hand, they can bare risks of non-compliance and frauds, which cause expenses for the process owner. On the other hand, revealing beneficial deviations helps to improve the process with innovative ideas. Regardless of these types, a process can only be protected or improved if deviations are detected.

Process deviations are entities in a process, mostly traces or events, that behave abnormal with regards to a baseline model. Without such a model, the task is more complex. Anomaly detection is the task of defining normality in a domain in conclusion with the declaration of any observation in the data which does not behave according to this normality. Anomalies are partitioned into three categories, that are point anomalies, contextual anomalies and collective anomalies [5].

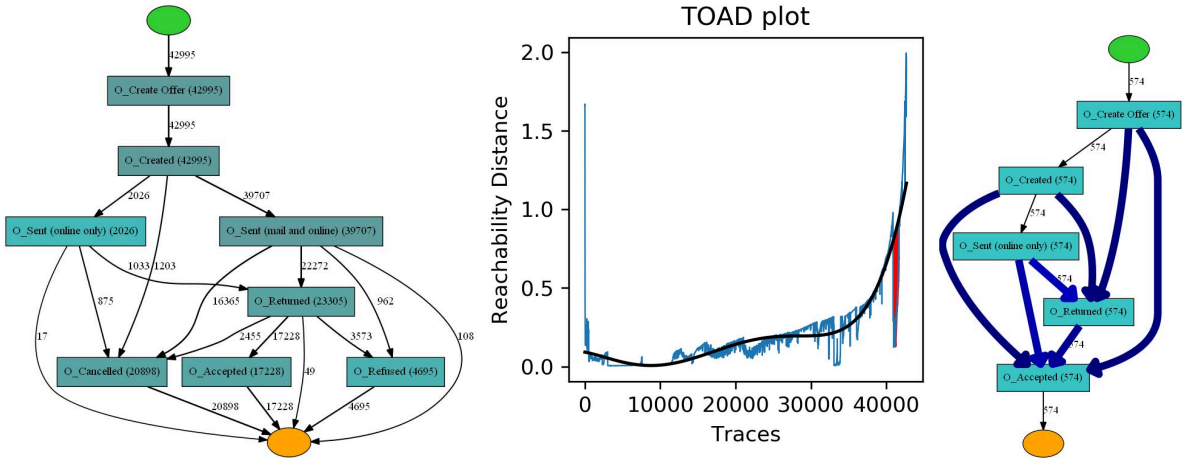
In traditional data mining, the search for point anomalies as outliers focuses on objects, that are prominently distinct in the sense of their behavior. The difference between an outlier and its most similar neighbor is typically very high [4]. This paradigm is useful in many scenarios, in which single objects have significant impact on the whole application. For example, a repair log contains reports from mechanics for a smart city bus fleet. If most buses succeed in a regular check-up, the ones that fail are point anomalies due to their individual failure. If such failures are depending on contextual factors, they call the anomaly a contextual anomaly. To continue with our example, some bus faults are season-sensitive. Air conditioning units, heaters or hydraulic systems are more likely to fail in summer and winter. Therefore, a faulty door in spring might have a different root cause than the same issue in winter.

However, in applications with extensive databases like process logs, where environments are continuously changing, it is not questionable if anomalies exists, but how to find them efficiently. Further, after the identification of anomalies, a solution for each issue is developed. This does not scale well for big data applications. In the end, it is a matter of efficiency and budget, which abnormal cases are treated and which have to be tolerated without any further actions.

In this work, we focus on collective anomalies. This paradigm ignores singular anomalies and yields clusters with abnormal behavior instead. The tagged abnormal traces might not be anomalies on their own, but their occurrence as a group is anomalous. Collective anomalies have been researched in the context of sequence data, spatial data and graph data. Especially for process data, this approach provides a strategy to efficiently handle anomalies from an economical perspective. Since it is very likely for an anomaly cluster that the root cause for the abnormality of all contained traces is the same, a common solution to the issue is applicable to the whole group. Returning to our example, a collective anomaly in the repair log reveals that some buses suffered from the same electrical short circuit, causing the hydraulic system to fail. Relying only on the previous paradigms, there would not have been an anomaly detected, since several buses had the same incident.

Our novel approach TOAD focuses on temporal deviations and utilizes temporal deviation profiles of cases, as presented in [16], to capture the temporal behavior of process instances. The concept of density is applied to define normality by the number of neighboring process instances. In [13], Tesseract showed that temporal outliers and drifts can occur even if the workflow of a process stays stable. However, Tesseract focuses only on drifts in singular activity relations. In this work, we extend the approach to the trace level and correlate activity relations of whole cases. Using the well-known cluster technique OPTICS on the temporal feature representation, we establish the name-giving Trace

978-1-7281-9832-3/20/$31.00 ©2020 IEEE
DOI 10.1109/ICPM49681.2020.00033



(a) Causal net for BPI Challenge 2017 Offer dataset. (b) TOAD plot with reference function. (c) Abnormal Submodel.

Figure 1. Process model of BPIC17 dataset, followed by reachability plot combined with a reference curve and the extracted anomaly process model.

Ordering for Anomaly Detection.

The main contributions of our work are (1) the adaptation of the temporal deviation signatures to generate a null hypothesis for an activity relation dependant density estimation, (2) transferring the concept of density to the process domain using OPTICS to identify micro-clusters of traces with higher relative density as anomalies. Further, we provide (3) a visual indication layer for process models to highlight and represent temporal deviation signatures.

2. Motivation

TOAD offers an explorative approach to identify collective temporal anomalies in process logs. Before we delve deeply into definitions and evaluations, we provide a real-world example to illustrate that collective temporal anomalies actually exist. The BPI Challenge 2017 dataset, which is described in detail in the evaluation, contains events of a load application process. In Fig. 1a we show the discovered causal net, yielded by the Heuristic Miner.

We apply TOAD, which identifies the most significant collective temporal anomaly. The yielded plot in Fig. 1b shows all traces on the x-axis, which are ordered according to their next neighbor, and the corresponding reachability distance, that is related to the local density of each trace. Therefore, a cluster of higher density is represented as a trough in this plot. The plot offers a simple and quick way to get an explorative view on the cluster structure in the dataset. We continue with automatically scanning for troughs with high relative troughs using a reference interpolation, plotted as a red line here. The troughs represent collections of traces, which are uncommonly more similar with respect to the expectation based on the remaining dataset. Depending on noise and the used parameters, many troughs are shown in the plot, however two very distinct ones are observed. We consider only the last one around $x = 40000$ due to its relative depth.

For its illustration, we mine a causal net only for the corresponding sublog in Fig. 1c. We add colored arrows as deviation indicators. The thick dark blue arrows show an accelerated execution of all events in this case in comparison to all other cases. These temporally abnormal cases contain only accepted loans. However, the delays indicate changed conditions in those cases, but we do not intend to theorize about the root cause here.

A deeper analysis is not possible without a domain expert and also not in the scope of this work. However, collective temporal anomalies exist in real-world process logs and due to their characteristics, traditional point anomaly detectors are not able to identify them. The model in Fig. 1c is a valid submodel of the original process model in Fig. 1b. Hence, anomaly detection based on the workflow would also miss this anomaly.

3. Related Work

In the area of clustering, especially trace clustering, some works have investigated different clustering techniques to aggregate process instances. The first established methods focus on vector space embeddings and map features of the sequences to points in a vector space. Basic approachs use bag-of-activities embeddings, which ignore the sequential information and represent traces as frequency vectors. Attached attributes like resources or lifecycle data can also be embedded into the vector space. This allows to use the full range of clusterings operating on vector spaces like k-means.

The vector space embedding of traces was first explored by Greco et al. [10]. Song et al. [15] used trace profiles which extend the previous idea and allow additional data to be captured. The number of attributes is not restricted such that complex behavior can be clustered using high-dimensional feature vectors. Ferreira et al. [9] clustered traces by building first-order Markov models for each cluster with the expectation-maximization technique. In [2] Bose

et al. introduced a specialized edit distance on traces and proposed an agglomerative hierarchical clustering approach. They also developed an approach [3] using subtraces of arbitrary length instead of n-grams to represent traces as feature vectors.

These techniques usually yield a partitioning of traces. Identified clusters provide a high intra-similarity and a low inter-similarity. These techniques can be summarized as passive trace clustering. Active trace clustering approaches, which were firstly introduced by De Weerdt [7] as ActiTraC, use an active learning process to develop trace clusters by taking the process discovery perspective into account. Traces are not mapped into a vector space but are greedily aggregated to clusters until the fitness of the model for this cluster drops. Sun et al. [17] also followed the hierarchical clustering method with compound trace clustering CTC. They developed a top-down trace clustering splitting logs in sublogs such that the complexity of each sublog matches the average complexity. The log with the highest model complexity is then split again. k-process [14] was developed by Richter et al. and uses process discovery and conformance checking in a k-means like procedure. Trace cluster centroids are built as process models and the assignment is performed using conformance checking as a distance to determine the most fitting cluster.

All these approaches work well to identify prominent process variants. However, we do not aim at finding trace clusters that exist on a common level. The deviations that our novel method reveals, are not mature clusters but anomalies. They can eventually grow into larger subprocesses and proper variants as well, but in the current state, the presented methods do not detect the minor deviations.

In contrast to clustering, traditional outlier detection methods focus on the identification of singular objects that deviate from the majority. The motivation for anomalous singleton traces is very intuitive: Revealing traces that do not behave as demanded, resulting from fraud or application failures. There is a manifold of different classification methods to solve this task in different applications. We can not cover the whole field, but few recent techniques shall be mentioned. Nolle et al. [12] utilized autoencoders to detect anomalies in business processes by training the process behavior on potentially noisy datasets and without prior knowledge. In [11], Myers et al. propose a method to identify anomalous behavior and cyber-attacks in industrial control system logs using conformance checking. Darmawan et al. [6] developed a method to filter a log, such that outliers are discarded and discovery can be applied on a clean anomaly-free database. All those methods amongst many others put emphasis on singular outliers. However, our novel approach aims at anomalous micro-clusters, so it is not a direct competitor for those methods. Our method is, as a result of its design, not able to detect those singularities and a direct comparison is not suitable. To the best of our knowledge, the perspective we take with our novel method, has not been investigated before in the field of process mining.

4. Preliminaries

Now, we will give the baseline definitions we are using regarding process mining and density-based clustering. Starting with the resources, the input data for TOAD are process event logs. An event is an observed witness of a process action. It contains the information about what activity happened at which timestamp in which case context at least, but it can provide additional data attributes. Here, we refer to the set of possible activities by A. An event $e = (c, a, t) \in (\mathbb{N} \times A \times \mathbb{N})$ is a tuple consisting of a case identifier c, an activity a and a timestamp t. Each event describes the execution of a certain activity corresponding to the case c at time t. We call a multiset, that contains multiple events over the same activity domain, a log L. Semantically, an event is a unique occurrence of an activity. Nevertheless, it can happen that the same case contains repetitions of a particular activity. If such events occur consecutively in a very small time frame, they might share the same event representation. Further, we call tuples of two activities $(a_1, a_2) \in A^2$ a relation.

A case c contains only the subset of a log, such that events in this subset share the same case identifier. As case identifier and cases themselves represent the same object, we will use it bilaterally. A trace is the projection of a case to the sequence of activity labels. However, in the literature, both terms are not used decisively and are often used in mixed contexts. Since our method puts a strong focus on the temporal view, it should be kept in mind, that we always consider the timestamps as a part of a trace in this work.

Continuing with density-based structuring, we require a measure of density. As a brief reminder, a distance is a positive-definite function, that is symmetrical and fulfills the triangle inequality. We do not go into detail here and for all mentions of a distance in the following, we are using the Euclidean distance, if not stated otherwise.

Density-based clustering is based on the concept of density, which is defined by two parameters: The radius ε to define the neighborhood of each object $N_\varepsilon(o)$ and the minimal number of points $MinPts$ required for a dense neighborhood.

One of the most popular density-based methods is DBSCAN [8]. It selects points and classifies them depending of their neighborhood as core, border or noise points. For a more in-depth description, we point to the corresponding work of Ester et al. A major drawback of DBSCAN is the difficulty to choose an appropriate value for the neighborhood distance ε. To overcome this issue, Ankerst et al. developed OPTICS [1]. Given $MinPts$, this method determines for each point its core distance, the minimal distance needed such that the ε-neighborhood contains $MinPts$ many points. For a point p let kNN be the k-th nearest neighbor and d a distance function. With $k = MinPts$, the core distance is defined by

$$\text{core}_{\varepsilon, MinPts}(p) = \begin{cases} d(p, \text{kNN}), & |N_\varepsilon(p)| \geq \text{MinPts} \\ \text{undefined}, & \text{otherwise} \end{cases}$$

The ε value is used as an upper bound for performance improvement. Using the core distance, the reachability distance for two objects o, p is defined as

$$\text{reach}_{\varepsilon,MinPts}(o,p) = \begin{cases} max(\text{core}_{\varepsilon,MinPts}(p), d(p,o)), & \text{if } |N_\varepsilon(p)| \geq \text{MinPts} \\ \text{undefined}, & \text{otherwise} \end{cases}$$

The set of data points gets ordered by its reachability distance. For each point, its successor is the point with the smallest reachability distance out of the unprocessed points. This ordering is not unique, due to start point ambiguity and potential choices between equidistant objects. Finally, a reachability plot is provided using the ordering on the x-axis and the reachability distance on the y-axis. Since dense object clusters in the data space have low pairwise reachability distances, they are accumulated in the plot and the cluster is identified as a trough in the reachability plot.

For a clustering analysis, the OPTICS plot is used to determine an ε-level, such that as many troughs as one likes are separated under a horizontal line. With this technique, clusters of equal densities can be identified in an exploratory task. However, our goal is different. The result of our novel method does not reveal distinct trace clusters of similar densities. We provide small clusters with anomalous densities relative to their neighborhoods as a result. In the following section, we go into detail to describe our changes to the adapted clustering technique, such that it becomes suitable to identify the desired results.

5. TOAD: Trace Ordering for Anomaly Detection

Our novel algorithm TOAD comprises of three major steps to provide anomalous outlier micro-clusters. First, we explain the mapping into a data space that keeps track of all temporal deviations. Second, we order the traces using OPTICS, based on nearest-neighbors. In the third part, we detect anomalies by analyzing the ordered plot and extract trace candidates with strong deviation characteristics.

5.1. Temporal Deviation Signatures

In [13], the authors focus on temporal concept drift detection regarding singular relations only. We adapt the idea of standardization via z-scoring, as it proposes a suitable representation of deviations. However, we extend the perspective from event level to traces.

Starting with a process log, we extract all event pairs sharing the same case identifier. Considering all relation candidates we determine the relation duration set RD for a given relation (a_1, a_2) as follows:

$$RD(a_1,a_2) = \{t_2 - t_1 \mid e_1, e_2 \in L \wedge e_1 = (c, a_1, t_1) \wedge e_2 = (c, a_2, t_2) \wedge t_1 < t_2\}$$

This set contains all durations between two particular activities, that are correlated in common process instances. We extract the arithmetic mean and standard deviation of all those relation duration sets, where $n_{(a_1,a_2)} = |RD(a_1, a_2)|$.

$$\mu_{(a_1,a_2)} = \frac{1}{n} \sum_{\delta_t \in RD(a_1,a_2)} \delta_t$$

$$\sigma_{(a_1,a_2)} = \sqrt{\frac{1}{n} \sum_{\delta_t \in RD(a_1,a_2)} (\delta_t - \mu_{(a_1,a_2)})^2}$$

In the next step we perform a z-scoring, which deals with unstable activities by down-weighting them in comparison to activities with small standard deviations. Z-scoring provides a standardization for the duration values, so we can align short-term and long-term activities in the same model. According to the work in [16], we derive temporal deviation signatures for each case. A temporal deviation signature is a matrix $TDS_{a_1,a_2} \in \mathbb{R}^{A \times A}$ that assigns the standardized duration to the activity relation, if the activity relation is present in the case:

$$(TDS_c)_{a_1,a_2} = \frac{|t_2 - t_1| - \mu_{(a_1,a_2)}}{\sigma_{(a_1,a_2)}}$$

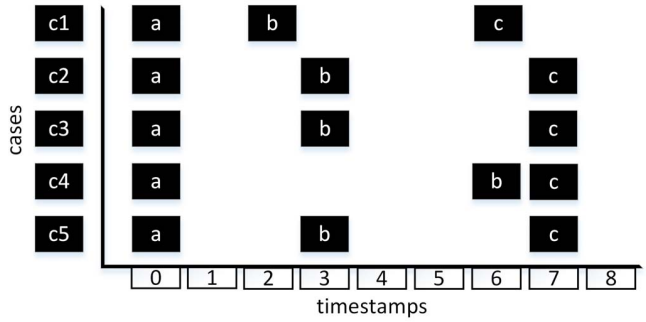
Otherwise, the value 0 is assigned.

For ordinary cases, which behave completely in-control, the deviation scores in the signature are next to zero. If we treat the signature as a vector in the linear space $\mathbb{R}^{|A|^2}$, those cases are represented as points around the origin, forming a sphere. Depending on the noise, the density of the sphere would follow the same distribution. For small datasets with few cases, Poisson-like characteristics towards the positive quadrants are likely, whereas larger datasets tend to form Gaussian-distributed density levels due to the central limit theorem.

We take a look at the example sketched in Fig. 2. All cases contain the three activities a, b and c and the majority require 3 time units between a and b and 4 time units between b and c. The first case $c1$ has a quicker execution and is therefore a deviation. The corresponding trace deviation signature is shown in Fig. 2b. The fourth case $c4$ shows also a deviation, as activity b is delayed, while the complete case execution stays close to the mean execution time.

5.2. Ordering Traces

In traditional data mining, a large manifold of clustering techniques have been proposed. We utilize density-based clustering as the next analysis step and adapt OPTICS [1] to perform the actual clustering. As we have explained before, ordinary cases will likely establish a sphere around the origin, if we treat the signatures as vectors. The maximum density is expected to be in the center. The more a singular case is deviating, the more it drifts towards sparser areas in the vector space representation. More interestingly, if a group of traces accumulates at a certain point, it establishes a denser area within sparsity. In the following we will show



(a) Traces of an example process, represented as Gantt chart.

	a	b	c
a	0.0	-1.0	-2.0
b	0.0	0.0	0.5
c	0.0	0.0	0.0

(b) Temporal deviation signature of case c1.

	a	b	c
a	0.0	1.9	0.5
b	0.0	0.0	-2.0
c	0.0	0.0	0.0

(c) Temporal deviation signature of case c4.

Figure 2. A tiny toy example demonstrating the derivation of trace deviation signatures.

how to automatically identify those collective anomalies or rare patterns.

DBSCAN [8] is a popular starting point for density-based clustering. It identifies clusters with similar densities and arbitrary shapes in the data space and can cope with noise. However, the density parameter is fixed so it can only identify clusters above a certain threshold level. Our data space varies in density and the more the temporal deviation signatures expand from the center, the sparser the anomalies get. The definition of anomalies does not rely on an absolute threshold, but on the surrounding space and its relative dependency gradient. As an extension to DBSCAN, OPTICS applies different levels of densities and creates a hierarchy of clusterings for a whole range of densities. The result is a 2-dimensional plot, that shows for each object, given on the x-axis, the density threshold necessary to be a cluster member. Since objects are ordered according to nearest neighbor distance, troughs in the plot depict dense aggregations of objects.

We establish a meaning of distance between two temporal deviation signatures S_{c_1} and S_{c_2}. Regarding distance measures, there is always the opportunity to dive deep into sophisticated measures. We will not focus on that here, but use the Euclidean distance. For two signatures S_{c_1} and S_{c_2}, the distance $\delta(S_{c_1}, S_{c_2})$ is defined as follows:

$$\delta(S_{c_1}, S_{c_2}) = \sqrt{\sum_{(a_1,a_2)\in A^2} (S_{c_1} - S_{c_2})^2}$$

Density is the central cluster criteria in this paradigm. It is basically the quotient of the number of traces in a certain area. For a signature S_c, we specify a distance threshold ε and count the number of other signatures $S_{c'}$ with a distance $\delta(S_c, S_{c'}) \leq \varepsilon$. The area within ε is called ε-neighborhood $N_\varepsilon(S_c)$. S_c is called dense or a core point, if its neighborhood contains more than $MinPts$ many other signatures for the predefined minimal point threshold $MinPts$. In other words, the distance for a neighborhood necessary to make a particular signature a core signature is given by

$$\text{core-dist}_{MinPts}(S_{c_0}) = \delta(S_{c_0}, S_{c_{MinPts}})$$

for the rising chain $\delta(S_{c_0}, S_{c_1}) < ... < \delta(S_{c_0}, S_{c_{|L|-1}})$. The definition is sound under the assumption that $MinPts \leq |L|$. Otherwise, the core-distance is undefined. Using the core-distance, we define the reachability distance between two signatures S_c and $S_{c'}$. This is either the core distance or the actual distance, whichever is greater:

$$\text{reach-dist}_{MinPts}(S_c, S_{c'}) = \max(\text{core-dist}_{MinPts}(S_c), \delta(S_c, S_{c'}))$$

Now, OPTICS orders all signatures. Starting with an arbitrary signature, all signatures are processed such that in each step, the signature with the smallest reachability distance to any previously processed signature is chosen as the next item. This way, closest points are processed with higher priority and compose a consecutive sequence in the process. The processing order defines the sequence of the final reachability distances to be selected, since only the distances between closest signatures are plotted. Hence, neighboring signatures are directly following in the reachability distance plot and the value of each signature defines the neighborhood size, such that both signatures belong to the same cluster. We will refer to this plot as the OPTICS plot.

5.3. Anomaly Detection

Traditionally, the OPTICS plot is used to visually identify clusters as valley-like structures. A density level is selected manually, such that the valleys under this level form valleys corresponding to the intuition. We propose an automated method, that does not reveal clusters with the same density, but anomalous aggregations of traces with deviating behavior instead. OPTICS applied to temporal deviation signatures usually yields a pattern, which assembles a large shallow valley for the majority of non-deviating traces, and which ascends into higher reachability distances while processing the deviating cases.

The OPTICS plot has indentations, that interrupt the otherwise smooth base-line of the analysis. The deeper such an indentation is, the denser is this small cluster of traces and the more similarity is shared between the affected traces. Hence, we compare the OPTICS plot with the smoothed base-line and focus on traces, that possess a large distance between smoothed and actual reachability distance to identify collective anomalies.

We generate the base-line by applying the Savitzky-Golay filter, a low-pass filter, to the reachability plot. It

can cope with the vast ascension in the outlier phase. Signal components of high frequencies are not discarded completely as in most other smoothing techniques. This advantage is achieved by using polynomial regression with low-degree polynomials over a small window around the particular point to be smoothed. The difference between original and smoothed curve is then used to identify deviations by applying peak-finding from the field of signal processing.

5.4. Limitations and Parameters

Now, we discuss some drawbacks of our method and their implications, as well as highlight the necessary parameters of TOAD. An important issue is the missing-of-values problem, which occurs by mapping traces to the temporal deviation profile space and apply a clustering method. Although all traces might be well-logged, traces usually contain only a subset of process-inherent activities. Hence, the temporal intervals are not defined for all relations in all traces. In our implementation, non-existing relations have a deviation score of zero. On the contrary, a zero is overloaded with the information, that the relation is present in the case, but does indeed show no deviation. Using the overloaded zero seems to be a naive but suitable solution. This causes a dense region at the zero, which is a false positive and can easily be discarded from the result set. Usually, anomalies that lack any abnormal behavior, are great to have but do not provide potential for improvement. However, we investigate other distance metrics in future works to find a better way for the comparison of 'incomparable' traces.

Regarding parameters, our method requires only few user-defined parameters. Since we adapt density-based clustering, the specification of density is a fundamental parameter. TOAD uses the number of minimum neighborhood points $MinPts$ to determine the trace ordering. For the resulting OPTICS plot, this parameter controls the granularity of the plot. A higher value increases the smoothness. We achieved good results with small values due to the following smoothing, e.g. $MinPts = 10$. After establishing the OPTICS plot, the Sawitzky-Golay filter needs the window size and the polynomial degree. As TOAD benefits from a very smooth baseline, we choose the window to be the maximum size possible. The polynomial degree should be an odd number. Usually the OPTICS plot starts low and raises when all denser points are processed and only singular outlier points are left. So for most distributions, the baseline is formed similar to a polynomial of degree 3 or 5. We suggest starting with those parameters. Later, the peak-finding requires to specify the width and prominence of a peak. Here, we suggest to use a top-k approach and start by analyzing high and large peaks first.

6. Evaluation

In the motivation section, we already show, that collective temporal anomalies actually exist. To the best of our knowledge, there are no other works that identify collective anomalies in the temporal perspective. So instead of a competitive analysis, we demonstrate the accuracy of TOAD by comparing results for different parameter settings and datasets. To conduct our experiments, we implemented TOAD as a stand-alone Python program that is available on Github[1].

6.1. Datasets

To evaluate the accuracy of an outlier detection method, problems for pure synthetic and real-world datasets arise. First, a synthetic dataset allows much freedom and it is very simple to design an artificial process that benefits the method to be evaluated. Anomaly detection has to be robust against noise. To show the robustness of TOAD, we need a dataset with an adequate level of realistic noise. It is difficult to claim that a synthetic dataset behaves realistically. Either the noise level is too low, which gives an advantage to the anomaly detection, or the noise is too strong and masks all anomalies. Therefore, a real-world dataset is a more suitable baseline, since the data obviously behaves realistically. However, we are mostly lacking a ground truth about which traces are anomalous. To overcome those issues, we mutate a real-world dataset by changing the duration of one activity. This simulates a group of traces, that might be more complicated and need additional time to process. We only mutate one activity at a time since more mutations make the abnormal cluster more distinct and simplify the detection.

We use the BPI challenge datasets from 2015[2] and 2017[3], in the following abbreviated as BPIC15 and BPIC17. BPIC15 contains data of building permit applications over four years in five Dutch municipalities. Five log partitions show the process of each municipality individually. Each sublog contains about thousand cases. The challenge of this dataset is not its number of cases. About 400 activities and therefore a large number of potential relations in all cases raise the complexity of the data.

In BPIC17, a loan application process of a Dutch financial institute over one year is logged. The offer log contains only a subset of 24 offer related activities. 128985 events are recorded in 42995 cases.

6.2. Experimental Setup

BPIC2015 contains a large activity domain. We identified the activity *phase decision sent* resp. *01_HOOFD_515* as a mandatory activity, that exists in every case. Although, this property is not a requirement for TOAD, it gives the opportunity to simulate a realistic and at the same time controllable deviation in all cases. We added a delay to this activity execution. The alteration of only one activity allows to filter infrequent activities and observe the impact of filtering while the detection challenge remains high.

1. https://github.com/Skarvir/TOAD
2. https://doi.org/10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1
3. https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b

Alterating multiple instances would simplify the detection as the distance to normal case profiles will rise. Since the remaining timestamps are unaltered, realistic noise still influences our results. In addition, other collective anomalies still exist in the data as well, so the result plot does contain further deviations.

For the BPIC2017 dataset, we manually identified a subset of cases which differ in their temporal execution. As already illustrated in Fig. 1c, this anomaly concerns accepted loan offers with an accelerated acceptance completion. The abnormal cases need on average 2.3 days to complete with a standard deviation of 1.9 days. In comparison, the remaining cases were completed in 13.5 days with 9.5 days as standard deviation. We applied TOAD on the first 5000 cases with different settings for MinPts. In addition, we used three different versions of the dataset. Version 1 contains all 5000 original cases. Version 2 contains only cases of length greater than 4 and Version 3 contains only accepted offers.

6.3. Results

Starting with the BPIC2017 dataset, we show in Fig. 3 the fitness and precision for the detection task to identify a collective anomaly as a classification problem. The measures should not be confused with process model quality, as it refers to classification accuracy. The anomaly consists of 135 cases, that represent accepted offers, but were more quickly accepted than on average. Roughly 1.5 standard deviations separates the abnormal cases from the remaining ones. In this setup, we compare the impact of the very important $MinPts$ parameter. For all three dataset variants, there is a clear drop in the fitness for $MinPts > 40$. Obviously, this value has to be smaller than the anomaly, that has to be detected. By choosing greater values, many smaller anomalies vanish in the result. However, the number of observed relations and therefore the number of dimensions increases distances between traces, so we do not detect the anomaly with a minimum of at least 40 traces within a cluster neighborhood.

The fitness drops slightly over the observed $MinPts$ values. This is explained by the fact that higher $MinPts$ values cause a smoother reachability curve and traces at the edges of a valley are not detected. The fitness for all three variants behaves similarly for increasing $MinPts$. For precision, matters change, as many cases contain only few activities and pairs of shorter cases tend to have smaller distances. Pruning of these short cases increases overall precision. As a rule of thumb, a narrow interval of case lengths is beneficial for the anomaly detection, as the dataset is more homogeneous. Otherwise, cases of short length are abnormal by size and are often detected first. The difference between all cases with length greater than 4 and only accepted cases is marginal, since the temporal representations differ in multiple dimensions. In the representation space, large traces exist in sparser areas and detection is made easier.

An important detail about the evaluation is the inequality of precision and fitness. Although the fitness drops significantly, the result is still very useful. We will illustrate this by applying TOAD on the modified BPIC2015 dataset. We inject abnormal traces by delaying the *phase decision sent* by a certain number of hours. On average, this activity follows its preceding event after about 6 days with a standard deviation of around 1 day. Due to the already existing temporal variance, the anomaly requires a delay of at least 8 hours to be detected. The precision is perfect for greater delays. We projected the log onto fewer activities, so the number of dimensions for the representation is reduced. Less dimensions increase the accuracy, which is not surprising. In the smallest case, only 6 activities are observed, leading to 30 relations, The third instance contains 11 activities, resulting in 107 relations. This dataset contains many unique activities or infrequent activities and a preprocessing for pruning is highly recommended.

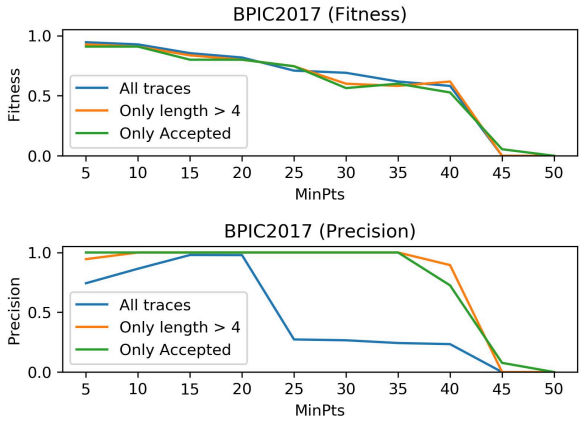


Figure 3. Accuracy scores of TOAD applied to the BPIC2017 dataset.

The fitness here is also not convincing by simply considering the numbers. We need to take a look on the actual result in Fig. 4. This result shows one of the more difficult evaluation instances. The reachability distance is plotted in blue, the reference curve is red. We highlighted the detected abnormal traces as a green area between the curves. All those detected traces belong to the ground truth, which leads to maximum precision. Taking a look onto the not detected abnormal traces, marked as yellow crosses, all but one trace are just besides the detected traces. TOAD is designed to identify a collective anomaly and the anomaly was correctly detected. Hence, the fitness value is not of the same importance as the precision value. Identifying and analyzing false anomalies costs much effort and time. Missing an abnormal trace is reasonable if the anomaly is detected. However, spending resources on chasing false anomalies should be avoided in many applications. If an actual detection algorithm for individual trace classification is needed, TOAD can still be used as a starting point and borders around the dense cores have to be investigated.

Summing up the results, the $MinPts$ parameter has to be chosen reasonable, since small values will lead to anomalies with few traces, while larger values cause most anomalies to disappear in the result. In the field of density-

based clustering, the research for good heuristics is still active. Also, pruning of short traces and narrowing the search space is very efficient.

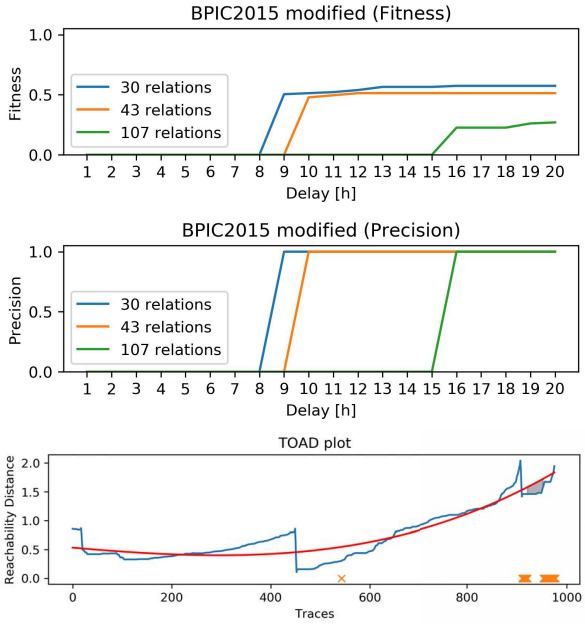


Figure 4. Accuracy scores of TOAD applied to the BPIC2015 dataset and TOAD result for the modified BPIC15, pruned to 107 relations with a 20h delay.

7. Conclusion

Identifying outliers is an important task for process improvement and failure prevention. In this work, we presented a method to detect micro-clusters with outlier characteristics. These process deviations as sublogs bear an economically efficient potential to search for process adaptations, either to include positive behavior or to avoid negative failures. Singular outliers are excluded in our method by design, as they require individual treatment and it is likely that their behavior does not occur a second time. Instead, our novel method TOAD puts emphasis on sets of traces, that possess a dense intra-density while being distant to other traces. Requiring only few parameters to specify and a low computational complexity, our method is well-suited for industrial applications.

In future works, we are planning to investigate the event stream capabilities of TOAD. A fast detection mechanism can help to detect outliers on-the-fly and call the attention of analysts to anomalies with a high impact due to the number of affected traces. In an operational support setting, the benefits of TOAD are even more beneficial, as analysis time becomes a scarce resource.

A further interesting direction adapts our approach and takes other attributes besides the temporal data into account. Weighting attributes of different specifications against each other like resources and time is a difficult task and more development is required to generalize TOAD to broader sets of data types. However, this branch is very yielding, because all sources of deviations are essential for a thorough root-cause analysis on micro-clusters.

References

[1] Ankerst, M., Breunig, M.M., Kriegel, H.P., Sander, J.: Optics: ordering points to identify the clustering structure. ACM Sigmod record **28**(2), 49–60 (1999)

[2] Bose, R.J.C., Van der Aalst, W.M.: Context aware trace clustering: Towards improving process mining results. In: Proceedings of the 2009 SIAM International Conference on Data Mining. pp. 401–412. SIAM (2009)

[3] Bose, R.J.C., van der Aalst, W.M.: Trace clustering based on conserved patterns: Towards achieving better process models. In: International Conference on Business Process Management. pp. 170–181. Springer (2009)

[4] Breunig, M.M., Kriegel, H.P., Ng, R.T., Sander, J.: Lof: identifying density-based local outliers. In: Proceedings of the 2000 ACM SIGMOD international conference on Management of data. pp. 93–104 (2000)

[5] Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM computing surveys (CSUR) **41**(3), 1–58 (2009)

[6] Darmawan, H., Sarno, R., Ahmadiyah, A.S., Sungkono, K.R., Wahyuni, C.S.: Anomaly detection based on control-flow pattern of parallel business processes. Telecommunication, Computing, Electronics and Control (TELKOMNIKA) **16**(6), 2808–2815 (2018)

[7] De Weerdt, J., vanden Broucke, S., Vanthienen, J., Baesens, B.: Active trace clustering for improved process discovery. IEEE Transactions on Knowledge and Data Engineering **25**(12), 2708–2720 (2013)

[8] Ester, M., Kriegel, H.P., Sander, J., Xu, X., et al.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Kdd. vol. 96-34, pp. 226–231 (1996)

[9] Ferreira, D., Zacarias, M., Malheiros, M., Ferreira, P.: Approaching process mining with sequence clustering: Experiments and findings. In: International Conference on Business Process Management. pp. 360–374. Springer (2007)

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

[11] Myers, D., Suriadi, S., Radke, K., Foo, E.: Anomaly detection for industrial control systems using process mining. Computers & Security **78**, 103–125 (2018)

[12] Nolle, T., Luettgen, S., Seeliger, A., Mühlhäuser, M.: Analyzing business process anomalies using autoencoders. Machine Learning **107**(11), 1875–1893 (2018)

[13] Richter, F., Seidl, T.: Tesseract: Time-drifts in event streams using series of evolving rolling averages of completion times. In: International Conference on Business Process Management. pp. 289–305. Springer (2017)

[14] Richter, F., Wahl, F., Sydorova, A., Seidl, T.: k-process: Model-conformance-based clustering of process instances. In: Proceedings of the Conference on "Lernen, Wissen, Daten, Analysen", Berlin, Germany, 2019. vol. 2454, pp. 161–172 (2019)

[15] Song, M., Günther, C.W., Van der Aalst, W.M.: Trace clustering in process mining. In: International Conference on Business Process Management. pp. 109–120. Springer (2008)

[16] Sontheim, J., Richter, F., Seidl, T.: Temporal deviations on event sequences. In: LWDA (2019)

[17] Sun, Y., Bauer, B., Weidlich, M.: Compound trace clustering to generate accurate and simple sub-process models. In: International Conference on Service-Oriented Computing. pp. 175–190. Springer (2017)

A Generic Framework for Trace Clustering in Process Mining

Fareed Zandkarimi
Chair of Enterprise Systems
University of Mannheim
Mannheim, Germany
zandkarimi@uni-mannheim.de

Jana-Rebecca Rehse
Management Analytics Center
University of Mannheim
Mannheim, Germany
rehse@uni-mannheim.de

Pouya Soudmand
Department of Industrial Engineering
Amirkabir University
Tehran, Iran
pouya.soudmand@gmail.com

Hartmut Hoehle
Chair of Enterprise Systems
University of Mannheim
Mannheim, Germany
hoehle@uni-mannheim.de

***Abstract*— The goal of process discovery is to visualize event log data as a process model. In reality, however, these models are often highly complex. Process trace clustering is a well-studied and powerful technique to address this. It groups an event log into more cohesive sub logs, such that the discovered process models become less complex and easier to understand. Over the past 15 years, researchers proposed various approaches for trace clustering in process discovery. The developed approaches vary greatly with regard to algorithmic capacities, data characteristics, computational complexity, and integration of additional information. In this paper, we provide a state-of-the-art analysis of trace clustering by a) performing a systematic literature review, and b) proposing a generic framework for trace clustering. Eventually, our goal is to provide an overview of current trace clustering research and a basis for developing new methods and approaches to trace clustering.**

Keywords*— *process mining, trace clustering, literature review, event log processing

I. Introduction

Process discovery is one of the three main functions in process mining (PM). Its main goal is to visualize a real-life business process, as recorded in an event log, in a human-readable way [1]. Therefore, discovery approaches generate process models, which are supposed to provide their users with a graphical representation of the examined process. In reality, however, these approaches often produce so-called spaghetti models, i.e., highly complex models that are difficult to read and understand [2]. This is particularly relevant because process discovery is typically the first step in a process mining project and the resulting models are used for many purposes, such as conformance checking or process simulation. Hence, improved process discovery results may lead to an overall improvement of process-based decisions made by humans and/or machines.

One way to achieve those improved results is to preprocess the event log before applying process discovery in order to decrease the process model complexity. For this purpose, process trace clustering is a well-known and effective technique [3]. It groups the traces in the log, such that the traces in each group (called cluster) are more similar to each other than to those outside the group, keeping the clusters as distinct as possible [4]. Applying process discovery on those more cohesive sub logs results in less complex and better readable models, which do not represent the entire event log at once. For example, the log of a hospital emergency room will contain multiple process variants, depending on the urgency, diagnosis, and treatment of the individual patients [5]. In order to analyze the process effectively, trace clustering can divide the log along with these attributes, such that the process flow for each group of patients can be examined individually.

Over the past 15 years, researchers proposed various approaches to apply trace clustering in the context of process discovery. The developed approaches vary greatly with regard to algorithmic capacities (e.g., density-based or hierarchical clustering), similarity functions (e.g., activity-based similarity), data characteristics (e.g., event log attributes), computational complexity, and integration of external knowledge (e.g., domain expertise). This makes trace clustering a context-specific task, i.e., the choice of the best approach depends on the availability of data, quality of the data, or available processing power, among others.

In this contribution, we aim to structure the existing body of knowledge in trace clustering. Therefore, we perform a systematic literature review to identify and summarize the current state-of-the-art. Based on that, we propose a generic framework for defining and structuring relevant building blocks of trace clustering methods. The goal of the framework is twofold. It summarizes the current trace clustering techniques and methods in the form of a methodological model and it provides a framework for further developing the field of trace clustering by identifying similar techniques in the related data mining areas. Therefore, we first summarize the background of clustering in process mining in Sect. II. Sect. III describes our research method. The results of our literature review and the proposed framework are covered in Sect. IV. Sect. V. discusses the results and concludes the paper.

II. Background

Cluster analysis is a well-established data mining technique aimed at finding groups of similar items in a dataset [6]. It is typically categorized as an unsupervised machine learning technique [7]. Clustering techniques are widely used for analyzing sequential data, e.g., sequence analysis techniques in bioinformatics, sequential pattern mining for user buying patterns, and sequence labeling for part of speech tagging. In the PM context, it can be applied to many different data objects, including events, activities, and process models. Arguably, however, the most relevant application is the application of clustering techniques on process event logs, which we focus on in this contribution. Trace clustering is often used as a preprocessing technique to improve process discovery results.

978-1-7281-9832-3/20/$31.00 ©2020 IEEE
DOI 10.1109/ICPM49681.2020.00034

For example, it can help to find statistical outliers and reduce noise in an event log [8], [9], but it may also support predictive monitoring of business processes by finding predicates that a running instance will most likely fulfill [10].

We define event logs as collections of traces that represent the behavior of a business process and assume that each trace has one mandatory attribute (the case identifier) and consists of a sequence of events, denoting the execution of activities. Each event also has mandatory attribute (the activity identifier). We call case identifiers, activity identifiers, and the ordering of events in a trace the *control-flow perspective*. If traces or events have additional attributes, such as timestamps, cost, and resources, we address those as the *context perspective*.

Trace clustering as a preprocessing technique for process discovery has been researched for more than a decade [11] and has experienced multiple developments regarding the techniques, similarity measures, computational complexity, and maturity. The morphological box by Thaler et al. gives a good overview of the approaches published before 2015 [12]. Early approaches to trace clustering treated traces as bags of activities, losing information on context or execution order [2], [13]. This was addressed by using different similarity measures such as the generic edit distance [3], sequences [14], or temporal proximity [15]. All of those approaches are two-staged, such that the discovery results are not considered during clustering. This problem is addressed by considering the properties of the discovered model during clustering [16], mining more accurate process variants or sub-processes [17], or finding a more appropriate distance measure [18].

Given the plethora of technically mature approaches, current research on trace clustering focuses on making the results more accessible for process analysts. De Koninck et al. describe an approach for explaining the assignment of traces to clusters [19] and a new technique for trace clustering that incorporates expert knowledge [20]. Seeliger et al. present the ProcessExplorer tool, which is set out to support the typical workflow of a process analyst to interactively explore a dataset [21].

Although a large body of research on trace clustering already exists, we are currently unaware of any generic framework, which would help to systematically assess the body of knowledge and the gaps in the field.

III. Research Method

In order to design such a generic framework for trace clustering in process discovery, we first conducted a systematic literature review [22], summarized in Table I.

TABLE I. SYSTEMATIC LITERATURE REVIEW

Rd	Search Criteria	Time	Results
1	"clustering" + "process mining"	2000 - 2020	5860
2	("trace clustering") + ("process mining") ["trace clustering" clustering OR process OR processes OR traces "process mining]	2000 - 2020	691
3	English papers only	2000 - 2020	630
4	Relevance according to titles; removing case studies, literature reviews, books, and short papers	2006 - 2020	126
5	Relevance according to abstract and content, removing duplicates	2006 - 2020	70
6	Forward and backward search	2006 - 2020	103

We conducted our search using Google Scholar, which contains, among other databases, IEEE Xplore, SpringerLink, and the ACM Digital Library. Our final search string was designed to capture studies mentioning trace clustering and process mining. We started without any further filters in the first round, and then restrained the search terms and filters after each iteration. Keyword search inside the whole text for peer-reviewed publications in English returned 630 studies. We filtered out duplicates, books, short papers, datasets, and irrelevant studies (based on title, abstract, and keywords). The resulting set contained 126 studies, which we filtered based on their content. 70 papers were kept as relevant. We then conducted a forward and backward search on those papers to cover studies missed by term search [22]. Finally, a set of 103 papers was used to conduct this literature review[1]. Table II shows the distribution of papers over the years.

TABLE II. DISTRIBUTION OF THE COLLECTED STUDIES PER YEAR

Year	# Papers	Year	# Papers	Year	# Papers
2004-2011	16	2014	7	2018	11
2012	4	2015	11	2019	11
2013	7	2016	11	2020	7
		2017	18		

We used the 103 papers found in the literature review to design a generic framework for trace clustering in process mining. First, we identified common concepts and building blocks that constitute different trace clustering approaches. Then, we focused on the ordering and dependencies of these building blocks. This included explicating the data flow between them and differentiating between necessary and optional steps. At this point, we had a framework that illustrated the generic procedure of trace clustering as found in the current literature. We then extended this framework with additional building blocks that were not found in the process mining domain, but were established and promising techniques that are used for cluster analysis in data mining. The purpose of this step was to get a more complete overview of trace clustering and to show opportunities for future research. Finally, we went back to the 103 papers and identified the concepts and techniques that they used to realize each generic building block. This framework is a methodological model, as it is derived from studying trace clustering methods and reflects the possible method development paths. We also argue that this model is mostly descriptive (relying mostly on the given literature) and partly normative (due to introducing new building blocks).

[1] A detailed list of these studies is available at https://doi.org/10.6084/m9.figshare.12607742.v2

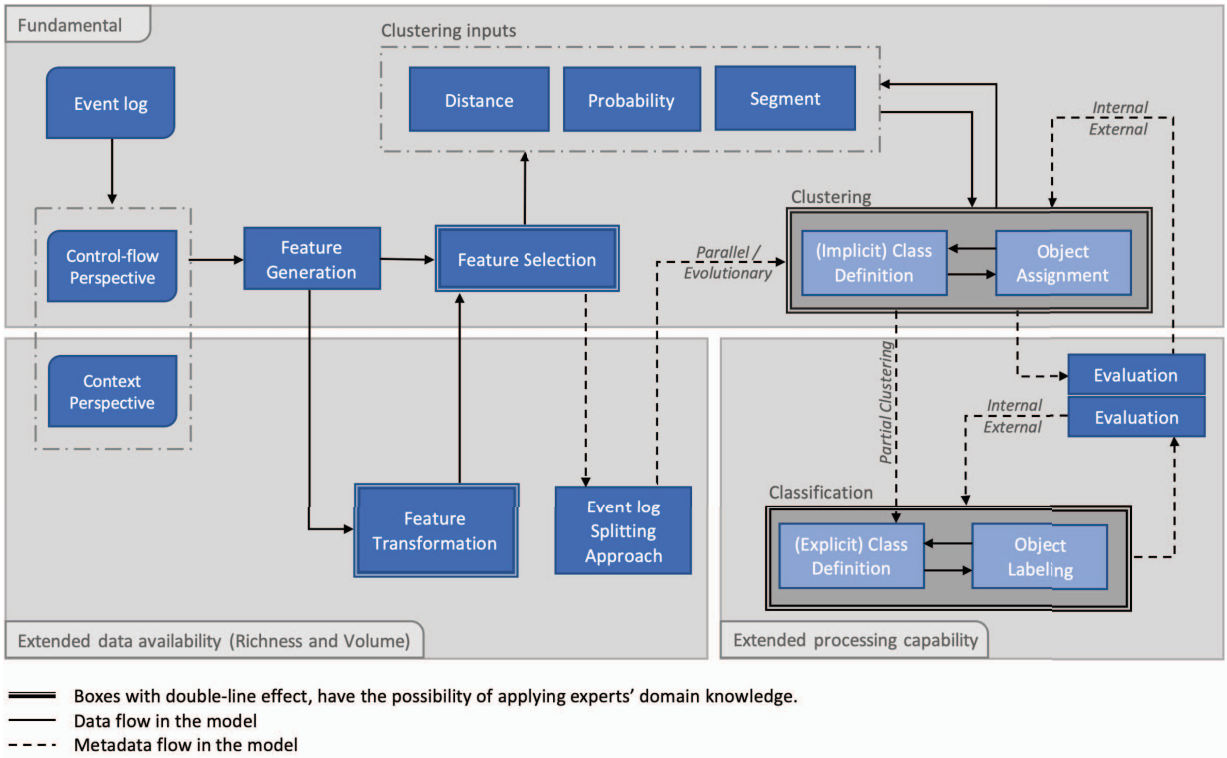


Fig. 1. Generic Framework for Trace Clustering in Process Mining

IV. A Framework for Trace Clustering

A. Overview

Figure 1 shows our proposed generic framework for trace clustering in process discovery. It consists of blue boxes, which describe the generic building blocks of trace clustering approaches. One of the first things we noticed in analyzing the trace clustering papers was that many of these building blocks depend on the availability of either certain attributes in the event log to measure trace similarity or certain processing capabilities to execute computationally complex clustering algorithms. Therefore, we divided the building blocks into three major groups, indicated by the grey areas in the framework. Building blocks in the "fundamental" area can be applied in any trace clustering context, where a basic event log and standard processing capabilities are present. Building blocks in the area "extended data availability" can only be applied if the event log fulfills certain properties regarding data richness and volume. This may refer to the presence of a certain attribute, the number of attributes, or the amount of traces in the log [23]. Finally, building blocks in the "extended processing capabilities" area can only be applied if large computational resources are available, because standard capabilities are not able to apply computationally complex algorithms to large event logs.

The building blocks of the framework are connected by arrows, which represent their ordering in the trace clustering process and the flow of data between them. The solid-line arrows show the flow of event log data, while the dotted arrows are signs of exchanging metadata, e.g., signals or parameters.

We consider domain knowledge as an optional input to trace clustering, which can be utilized to different building blocks. These blocks are marked with a double-line effect meaning that during our literature analysis, we found the potential of applying domain knowledge in that particular step.

Each trace clustering project starts with an event log, which is represented by the building block in the top left corner. As explained in the background section, we consider two general perspectives on the event log, depending on the available data. The control-flow perspective only refers to case identifiers and sequences of activity identifiers (i.e., events). All possible paths in the fundamental area are applicable to such a basic event log. If a given event log contains additional attributes, e.g., resources, timestamps, or cost, these are represented by the context perspective. In this case, the blocks in the "extended data availability" area become applicable to this event log.

All available attributes in an event log can be mapped to either of the two following possible granularity levels: *activity-level* (e.g. cost of activity, department, user) and *trace-level* (e.g. order type, order duration, customer satisfaction). In terms of data types, all attributes in the event log are either *numeric* or *categoric* data. Transforming these two data types into each other occurs in the succeeding steps.

In the following sections, we discuss the framework's building blocks in more detail. A summary of the concepts and techniques used to realize them is presented in Table V.

B. Feature generation

The first step towards trace clustering is to generate features from the provided event log [24]. A feature can be any property or attribute contained in the event log. As the overall goal is to assess the similarity between individual traces, features will be generated at trace or event level. According to our observations in the literature, two *data structures* are typically used for feature generation. We categorize them as *linear* structures and *non-linear* structures. Linear features typically refer to simple data structures, such as scalars. Non-linear structures, e.g. graph and tree structures are used when the non-linear behavior of traces cannot be fully captured by those linear data structures. Depending on the required features, *numeric* or *categoric* data from the event log may get transformed from one type to another. In the following, we provide examples for such transformations.

Assume a sample trace <abac> with the following durations <15, 12, 8, 5> associated with the respective events. For this trace, we can generate a new feature with a different data type by mapping any duration value lower than 10 to low (L) and values higher than 10 to high (H), resulting in the (linear) feature <H, H, L, L>. This is a simple example of converting numeric data type to categoric.

TABLE III. TWO EXAMPLES OF TREATING CATEGORIC AND NUMERIC DATA TYPES

Input	Matrix	Vector	Scalar
$< abac >$	$\begin{matrix} a \\ b \\ c \end{matrix}\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}$	$\sqrt{6} \approx 2.45$
$< 15,12,8,5 >$	$\begin{matrix} a \\ b \\ c \end{matrix}\begin{bmatrix} 15 & 0 & 8 & 0 \\ 0 & 12 & 0 & 0 \\ 0 & 0 & 0 & 5 \end{bmatrix}$	$\begin{bmatrix} 23 \\ 12 \\ 5 \end{bmatrix}$	$\sqrt{698} \approx 26.42$

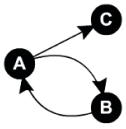
Table III provides an example of three possible representations of a trace <abac> with durations <15, 12, 8, 5>. This trace can be represented by a two-dimensional array (matrix), a one-dimensional array (vector), or a single value (scalar). Table III shows how those numeric features are extracted from a categoric (first row) and a numeric (second row) input. In both examples, the rows in the matrix represents the activities and the columns represent the positions in the trace, meaning that activity *a* appears both in the 1st and 3rd position and that the duration of the 2nd activity is 12. Depending on the required level of information abstraction, different transition functions (e.g., count, sum, mean) can be applied to transform these matrices into new *data structures*. In our example, the rows are summed up to transform the *matrix* into a *vector*. This reduces the feature size and therefore its sparsity, but also loses information on the trace ordering. The same happens when the vector is transformed into a scalar by applying a *magnitude* (sum of squares) function [15].

Moving from matrices to scalars results in a higher abstraction level. A similar observation would be moving from activity-level to trace-level features. The provided example in Table III shows individual activities of a trace, but one can consider the same features for any given subsets of a trace.

A subset of trace *t* is also known as an *n*-gram, where $0 < n \leq$ length(*t*). For example, <ac (0)> is a 2-grams of <abac> that represents the occurrence of *a* followed by *c* with a distance of zero. Two principal approaches can be considered when processing an event log based on *n*-grams: *manual* (brute force) and *algorithmic* approach (explorative algorithms) [13]. The manual approach tries all possible *n*-grams and extracts features from them, whereas in the algorithmic approach, only significant *n*-grams, such as the most frequent ones, are used for feature generation [25].

In our example, the underlying data structure is linear, i.e. we can interpret the trace as a sequence of activities that can be cut into subsets of different sizes. However, one could also conceptualize a trace as a graph structure, highlighting the existence of a link between any pairs of activities [26], [27]. In this case, we can generate new features by, e.g., discovering isomorphic subgraphs. As an example of the same trace (<abac>), the degree vector would be <2,1,1> for the respected undirected graph. For the directed graph, we can consider <3,2,1> as the degree vector, <2,1,0> as the outgoing vector, and <1,1,1> as the incoming vector (see Table IV).

TABLE IV. CONCEPTUALIZING A TRACE AS A GRAPH

Graph	Matrix	incoming	outgoing
	$\begin{matrix} \\ a \\ b \\ c \end{matrix}\begin{matrix} a & b & c \\ \end{matrix}\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{matrix} a \\ b \\ c \end{matrix}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$	$\begin{matrix} a \\ b \\ c \end{matrix}\begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}$

C. Feature transformation

The generated features of the previous step provide the input for the next one. Feature transformation includes generating secondary features and assuring feature quality and interpretability by applying techniques for normalization, collinearity, and dimensionality. Normalization ensures the comparability of features. Collinearity checks for redundancy, i.e., eliminating identical or similar features. Dimensionality ensures a certain degree of richness, so multiple versions of one attribute exist in the final pool of features, so if feature selection removes certain attributes, not all information is lost [28].

Secondary (or transformed) features are generated by applying *linear* techniques, e.g. wavelets [29], principal component analysis (PCA), factor analysis [30], or *non-linear* techniques (machine learning (ML)-based and non-ML-based).

D. Feature selection

In the last step of the pre-clustering phase, the generated and transformed features are reduced by feature selection techniques [11]. Those techniques are needed to avoid problems with overfitting, precision issues, and extra processing costs, which can appear when using too many

features. However, the feature selection process itself needs processing resources, too. It happens either in an *all-at-once* (single-view) or *incremental* (multiple-view) fashion [31]. The former usually yields relatively fewer features, whereas the latter allows for selecting more features in the final set.

Generally, feature selection techniques can be split into two categories, depending on whether they are based on machine learning (ML). *Non-ML-based* techniques include for example entropy-based filtering [16], filtering collinear features [32], and frequency-based selection. They also allow to integrate experts' domain knowledge into the feature selection process.

The *ML-based* techniques can be divided into two main approaches, *embedded* and *wrapping*. The embedded approach is a built-in feature selection mechanism that occurs during some trace classification algorithms [33]. It refers to the selection of significant features during trace classification, which we call partial classification. After processing a subset of traces, a set of clusters is defined, to which the remaining clusters are then assigned [10]. Features are justified based on the model performance, i.e., significant features have better accuracy in assigning clusters. The wrapping approach, on the other hand, tests different combinations of features in order to find the most significant ones. Features are justified based on the evaluation results of ML techniques [34].

E. Event log splitting approach

Dataset splitting is a common approach in data mining field mostly to handle complex and large data volumes [7], [35], which can become relevant for trace clustering as well. Dealing with large event logs can be computationally expensive and time-consuming. Processing the whole event log, however, does not guarantee optimal results. Hence, depending on the dataset, technique, and available computing resources, it might make sense to preprocess the event log by splitting it prior to clustering. Principal approaches in dealing with large event logs can be labeled as parallel and evolutionary. In both approaches, the event log is split into multiple sub logs.

The parallel approach exploits clustering and classification techniques to process all sub logs simultaneously. The applied settings and algorithms can vary for each sub log, to ensure the distinctness of the results. Processing each sub log yields a set of sub-clusters, which are merged to obtain the final clusters. In the evolutionary approach, randomly sampled sub logs are processed sequentially, until either all sub logs are processed or the updated clusters do not change significantly.

None of the approaches that were found in the literature review had implemented event log splitting prior to clustering. However, we included it in the framework, because it's an established technique in data mining, which we propose and recommend for future research, especially when dealing with very large event logs.

F. Clustering inputs

Clustering algorithms aim to group similar traces such that traces inside the same group are similar and traces from different clusters are different. The way in which trace (feature) similarities and differences are measured is therefore central to the resulting clusters. *Distance measures* are a popular technique for measuring trace similarity. Using them, we can compute a so-called similarity matrix, which contains pairwise similarity values for all traces in a log and serves as input for some clustering algorithms. Other potential inputs for clustering algorithms are *probabilistic models* and *segments*. These three potential building blocks of trace clustering are shortly discussed in the following.

1) Distance

There are multiple ways to calculate the distance between two traces (or their respective features), which vary according to *distance type*, *granularity*, and *robustness function*. The distance type can either be *string-based*, accepting all types of features, or *arithmetic*, accepting only numeric features. Calculating arithmetic distance requires mathematical operations which is not the case for string distance. Instead, *string-based* distance measures quantify the differences between two sequences of features of any type. One way to compute these differences is the *edit distance*, which refers to the number of operations (insert, delete or move) required to make two given sequences completely similar. In case of actual strings, this is also known as the Levenshtein distance [36]. Other string distance functions use different methods to calculate these differences. *Behavioral distance* is applied for features with a tree-like data structure. *Morphing-based* functions are used to compare traces. The *base-model distance* approach discovers an initial model, e.g. a graph, based on limited or all traces. This base-model calculates pairwise edge distances for respective nodes of the traces on the model and updates the results [27].

So far, we assumed that all approaches compare traces to model their differences. However, the same is possible by changing the unit of processing to batches of traces, i.e., segments [37]. Transition functions can build new "per segment" features. One last optional part of this step is implementing robustness functions. These functions help to moderate inappropriate distance values, happening due to the presence of erroneous data points (e.g. noise and outlier) in the event log [38].

2) Probability

Another option for trace clustering is to use probabilistic computation models instead of deterministic ones. In this approach, Markovian probabilistic models are built. The probability of observing a trace given the Markovian probabilistic model(s) determines to which cluster each trace belongs [39]–[41].

3) Segments

The third option in clustering inputs are segment-based computation techniques. In this approach, a conceptual *grid* is built, either gradually or all-at-once. In the all-at-once method, each segment together with its neighbors is tested to see whether merging them supports the goal of clustering or not [42]. The gradual model starts with the whole event log. In each iteration, the event log is split into segments until the clustering algorithm decides to stop. This decision is made based on the expressive power of clusters to satisfy the clustering goal. The final segments are basically the final clusters [43].

TABLE V. UNDERLYING TECHNIQUES AND CONCEPTS OF THE PROPOSED FRAMEWORK

Event log	**Feature Generation**	**Feature Transformation**	**Distance measure**
- Data availability: control-flow perspective (case identifier, activity identifier, order), context perspective (optional) - Data granularity: activity-level (e.g. duration, user, resources), trace-level (e.g. throughput time, quality, success) - Data type: categoric, numeric	- Data structure -- Linear: matrix, vector, scalar -- Non-linear: graph, tree - Data type: categorical, numerical - Transition function: magnitude, mean, max, frequency, etc. - Length: 1-gram, …, n-gram - N-gram building: manual, algorithmic - Granularity: activity-level, trace-level	*Linear* wavelet, PCA, factor analysis *Non-linear* - ML-based -- SOM, Deep belief Network - Non-ML-based -- kernel PCA, principal curves, Laplacian eigenmaps, diffusion maps	*Type* - String-based -- edit distance --- data structure: linear, graph --- distance function: Jaccard, hamming, levenshtein, etc. -- behavioral distance --- data structure: tree --- distance function: morphing -- base-model distance --- data structure: linear, graph --- distance function: geodesic - Arithmetic -- data structure: linear, graph -- distance function: Euclidean, cosine, google distance *Robustness function* - Moderation functions *Granularity* trace, segment
Clustering	**Evaluation**	**Feature selection**	
Input type - Distance-based hierarchical (bottom-up), density-based, partitioning - Probability-based - Segment-based grid-based clustering, hierarchical (top-down) *Assignment type:* hard, soft	- Feedbacking: internal, external - Criteria: PM-related (improving clustering), non-PM-related (business use cases)	*Mode* - Incremental, all at once *Type* - Non-ML-based expert-knowledge, entropy, frequency, collinearity - ML-based embedded, wrapping	

G. Clustering and classification

Clustering input blocks provide the necessary input data for clustering and classification algorithms. The *clustering* algorithm triggers new rounds of building the clustering input(s), iteratively. The provided input(s) are used by the clustering algorithm directly (e.g. k-means) or with further transformations (e.g. spectral clustering). We considered three main approaches for clustering, namely *full clustering*, *full classification*, and *partial clustering*. In the full clustering approach, detecting the initial clusters (implicit class definition) as well as assigning all traces to those clusters are all done mainly by the clustering algorithm. This approach also allows for integrating experts' knowledge when defining the initial clusters [5]. In the full classification approach, expert knowledge (in the form of manual grouping or to-be process models) is used to define the initial classes (explicit classes) that are later used by classification techniques to assign the rest of the traces to the clusters [44]. In the partial clustering approach, a combination of clustering and classification techniques is used. Clustering techniques, with the possible help of experts' knowledge, are applied to build the initial clusters based on a limited part of the event log. Eventually, classification part takes the final clusters as its initial classes and continues with processing the rest of the event log.

All three presented clustering inputs in the framework, i.e. distance models, probabilistic models, and segment-based models, provide input to feed classification and clustering algorithms (except segment block that cannot feed classification algorithms). Clustering building blocks and clustering techniques can be matched as follows. *Segment-based* matches with *grid-based* and *top-down hierarchical clustering* [2]. *Distance-based* matches with *hierarchical* (bottom-up) [45], *density-based* [46], and *partitioning* [38]. *Probability-based* clustering methods can be categorized based on their kernel methods (e.g. Gaussian, non-Gaussian) as well as their estimation methods (e.g. Bayesian, non-Bayesian).

In terms of cluster borders, clustering techniques divide into two categories, one allowing for overlapping (*soft*) and the other one not (*hard*). We observed implementing probability-based approaches for soft cluster splitting, however fuzzy approach is not implemented in any of observed trace clustering studies, so it remains as our suggestion for future research.

H. Evaluation

Usually classification and clustering algorithms have their own built-in objectives. However, we can also define extra objectives (evaluation criteria) depending on trace clustering's ultimate goal, e.g. enhancing process model discovery, demands for higher model simplicity, and fitness as evaluation objectives [12]. When a paper is not focused on making a methodical contribution to trace clustering or process mining, e.g., in case studies, usually standard built-in objectives suffice (we refer to this as non-PM-related evaluation criteria).

The purpose of defining extra objectives is assisting clustering and classification algorithms (PM-related evaluation criteria). This is achieved by generating feedback in two different ways, namely internally and externally. Internal feedback is generated per each iteration of running the algorithms (internal) [16], while external is generated once, i.e. after finishing the algorithm (external) [38].

Internal feedback improves the precision of assigning traces in each iteration of running classification and clustering algorithms. External feedback optimizes the parameters of classification and clustering algorithms, e.g. number of clusters, nominated set of features, weights of each building block, etc.

It worth mentioning that clustering and classification algorithms can have multiple objectives [47]. There are several techniques to handle this situation based on multi-objective decision making (MODM) approaches, e.g. weighted sum, disaggregation method, goal programming, and Pareto optimality.

I. *Demonstrating a sample study in the framework*

In order to demonstrate the validity of our framework, we briefly introduce an individual study (Delias et al [38]) and map its components to our proposed framework. The authors exclusively work with the control-flow perspective of the event log. In the feature generation step, for each trace, they generate two trace-level numerical features in the form of vectors. Two transition functions generate these vectors based on appearance of each activity in the trace (1-gram subsets) and pairwise inversed sequence-distance of activities (2-gram subsets). In terms of advanced feature generation, we did not observe any relevant actions. The authors applied a cosine similarity distance function on both available features to generate two similarity matrices, namely activities-similarity and transitions-similarity (we refer to as clustering inputs). In this approach, neither event log splitting nor classification techniques were implemented. Spectral clustering algorithm applied two different weights (determined by the help of experts' knowledge) to each similarity matrix and then combined them into a single matrix using weighted sum function. We observed exploiting a robustness function (density-based weighting) to reduce the effect of outlier and noises in the data. In the absence of PM-related goals, no evaluation (internal, external) loops were presented in this study.

V. CONCLUSION

Trace clustering has been a topic of interest in process mining research for almost two decades. In this context, the goal of this paper was to structure the existing body of knowledge and give a comprehensive overview on the state-of-the-art in trace clustering. Therefore, we first performed a systematic literature review, in which we identified 103 relevant research works on trace clustering between 2004 and 2020. We then used these works to design a generic framework on trace clustering in an iterative, bottom-up way. This framework consists of 15 building blocks, which are grouped by applicability. The flow of data and metadata between those building blocks represents the generic process of trace clustering. In a second step, we analyzed how the 103 identified research works realized some of those building blocks, leading to a conceptual and technical overview of trace clustering capabilities, as shown in Table V.

However, our research also suffers from multiple limitations. Despite following a methodical approach, we do not claim that our literature review or the framework are complete or exhaustive. Our choice of search terms or databases could have excluded relevant research works, which either used a different terminology or were not contained in the respective databases. Hence, there could be unidentified contributions to trace clustering, which could add additional aspects to our framework. The framework in its current form insinuates a generic process of trace clustering, which we have found to be true for many existing approaches, but which new approaches to trace clustering do not necessarily have to follow.

In addition, the framework's current building blocks result from our understanding and interpretation of the existing literature and could be conceptualized differently by other researchers. For example, one could argue that there are more than two perspectives to look at an event log or that trace classification is not part of trace clustering in a narrow sense. Also, our list of concepts and techniques that realize the respective building blocks can also not seen as complete, because it's highly likely that we missed or misclassified some approaches, for example due to different parametrization options. Our framework also falls short with regard to the generic data mining aspects and will require a more thorough argumentation on how they could address apparent gaps in trace clustering.

The overall goal of our generic framework is twofold. For practitioners, we want to give an overview over the current state-of-the-art in trace clustering and take a first step towards building the big picture of trace clustering techniques and implementation approaches. Categories like "extended data availability" and "extended processing availability" may help in this regard. However, the framework in its current form lacks practical utility, because it focuses more on the generic trace clustering process than on the capabilities of individual approaches. In addition, it has not been empirically validated by, e.g., other researchers or practitioners. In future work, this can be addressed by asking authors of our identified papers to position their work within our framework.

For researchers, we want to make it easier to position their own research with regard to the state-of-the-art in trace clustering. Although this is not the main focus of this paper, several of these gaps already became evident during our analysis. For example, there are no works on the "cluster-ability" of an event log, i.e., the attributes that an event log has to fulfill for trace clustering to be useful. Similarly, depending on the characteristics of the log, some clustering approaches could be more useful than others. These and many more challenges can be found from inspecting the state-of-the-art as represented in our framework and form a good basis for future research.

REFERENCES

[1] W. van der Aalst, Process Mining - Data Science in Action, 2nd ed. Berlin, heidelberg: Springer Verlag, 2016.

[2] A. K. A. de Medeiros et al., "Process Mining Based on Clustering: A Quest for Precision," in Business Process Management Workshops, LNCS 4928, A. ter Hofstede, B. Benatallah, and H.-Y. Paik, Eds. Berlin: Springer, 2008, pp. 17–29.

[3] R. P. Jagadeesh Chandra Bose and W. M. P. van der Aalst, "Context Aware Trace Clustering: Towards Improving Process Mining Results," Proceedings of the SIAM International Conference on Data Mining, SDM 2009. Sparks, Nevada, USA, pp. 401–412, 2009.

[4] H. A. Reijers, J. Mendling, and R. M. Dijkman, "Human and automatic modularizations of process models to enhance their comprehension," Inf. Syst., vol. 36, no. 5, pp. 881–897, 2011.

[5] X. Lu, S. A. Tabatabaei, M. Hoogendoorn, and H. A. Reijers, Trace Clustering on Very Large Event Data in Healthcare Using Frequent Sequence Patterns, vol. 11675 LNCS. Springer Verlag, 2019, pp. 198–215.

[6] B. S. Everitt, S. Landau, M. Leese, and D. Stahl, Cluster Analysis, 5th ed. Chichester, UK: Wiley, 2012.

[7] P. Berkhin, "A survey of clustering data mining techniques," in Grouping Multidimensional Data: Recent Advances in Clustering, Springer Berlin Heidelberg, 2006, pp. 25–71.

[8] A J. M. M. Weijters and W. M. van der Aalst, "Process mining: Discovering workflow models from event-based data," Proc. 13th Belgium-Netherlands Conf. Artif. Intell., no. i, pp. 283–290, 2001.

[9] M. F. Sani, W. Van Der Aalst, S. J. Van Zelst, and W. M. P. Van Der Aalst, "Applying Sequence Mining for Outlier Detection in Process

Mining: Data-Driven Value Proposition View project Process Querying View project Applying Sequence Mining for Outlier Detection in Process Mining," Springer, vol. 11230 LNCS, pp. 98–116, 2018

[10] C. Di Francescomarino, M. Dumas, F. M. Maggi, and I. Teinemaa, "Clustering-Based Predictive Process Monitoring," IEEE Trans. Serv. Comput., vol. 12, no. 6, pp. 896–909, 2019

[11] G. Greco, A. Guzzo, L. Pontieri, and D. Saccà, "Mining expressive process models by clusteringworkflow traces," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 3056, pp. 52–62, 2004

[12] T. Thaler, S. Ternis, P. Fettke, and P. Loos, "A Comparative Analysis of Process Instance Cluster Techniques," Proc. der 12. Int. Tagung Wirtschaftsinformatik (WI 2015), pp. 423–437, 2015.

[13] G. Greco, A. Guzzo, L. Pontieri, and D. Saccà, "Discovering expressive process models by clustering log traces," 2006. doi: 10.1109/TKDE.2006.123.

[14] G. M. Veiga and D. R. Ferreira, "Understanding Spaghetti Models with Sequence Clustering for ProM," in Business Process Management Workshops, vol. 43, S. Rinderle-Ma, S. Sadiq, and F. Leymann, Eds. Springer Berlin Heidelberg, 2010, pp. 92–103.

[15] D. Luengo and M. Sepúlveda, "Applying clustering in process mining to find different versions of a business process that changes over time," in Lecture Notes in Business Information Processing, 2012, vol. 99 LNBIP, no. PART 1, pp. 153–158,

[16] J. De Weerdt, S. vanden Broucke, J. Vanthienen, and B. Baesens, "Active Trace Clustering for Improved Process Discovery," Knowl. Data Eng. IEEE Trans., vol. 25, no. 12, pp. 2708–2720, 2013

[17] L. García-Bañuelos, M. Dumas, M. La Rosa, J. De Weerdt, and Chathura C Ekanayake, "Controlled automated discovery of collections of business process models," Inf. Syst., vol. 46, pp. 85–101, 2014.

[18] J. Evermann, T. Thaler, and P. Fettke, "Clustering traces using sequence alignment," Lect. Notes Bus. Inf. Process., vol. 256, pp. 179–190, 2016, doi: 10.1007/978-3-319-42887-1_15.

[19] P. De Koninck, J. De Weerdt, and S. K. L. M. vanden Broucke, Explaining clusterings of process instances, vol. 31, no. 3. Springer New York LLC, 2017, pp. 774–808.

[20] 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," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 10253 LNCS, pp. 561–576, 2017

[21] A. Seeliger, A. Sánchez Guinea, T. Nolle, and M. Mühlhäuser, "ProcessExplorer: Intelligent Process Mining Guidance," in Business Process Management, 2019, vol. 11675 LNCS, pp. 216–231

[22] J. Webster and R. T. Watson, "Analyzing the Past to Prepare for the Future: Writing a Literature Review," MIS Q., vol. 26, no. 2, pp. xiii–xxiii, 2002.

[23] M. Leyer, "Towards A Context-Aware Analysis Of Business Process Performance," PACIS 2011 Proc., Jul. 2011.

[24] M. Song, C. W. Günther, and W. M. P. P. Van der Aalst, Trace clustering in process mining, vol. 17 LNBIP. Berlin: Springer Verlag, 2009, pp. 109–120.

[25] R. P. J. C. Bose and W. M. P. Van Der Aalst, "Trace clustering based on conserved patterns: Towards achieving better process models," in Lecture Notes in Business Information Processing, 2010, vol. 43 LNBIP, pp. 170–181

[26] Q. T. Ha, H. N. Bui, and T. T. Nguyen, "A trace clustering solution based on using the distance graph model," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 9875 LNCS, pp. 313–322, 2016

[27] C. Diamantini, L. Genga, and D. Potena, "Behavioral process mining for unstructured processes," J. Intell. Inf. Syst., vol. 47, no. 1, pp. 5–32, Aug. 2016

[28] M. Song, H. Yang, S. H. Siadat, and M. Pechenizkiy, "A comparative study of dimensionality reduction techniques to enhance trace clustering performances," 2013

[29] F. Taymouri, M. La Rosa, and J. Carmona, "Business Process Variant Analysis Based on Mutual Fingerprints of Event Logs," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Jun. 2020, vol. 12127 LNCS, pp. 299–318

[30] E. Bartl, H. Rezankova, and L. Sobisek, "Comparison of classical dimensionality reduction methods with novel approach based on formal concept analysis," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2011, vol. 6954 LNAI, pp. 26–35

[31] A. Appice, D. M.-I. transactions on services, and undefined 2015, "A co-training strategy for multiple view clustering in process mining," ieeexplore.ieee.org.

[32] S. Lee, B. Kim, M. Huh, S. Cho, S. Park, and D. Lee, "Mining transportation logs for understanding the after-assembly block manufacturing process in the shipbuilding industry," Expert Syst. Appl., vol. 40, no. 1, pp. 83–95, Jan. 2013

[33] A. Cuzzocrea, F. Folino, M. Guarascio, and L. Pontieri, "Deviance-Aware Discovery of High-Quality Process Models," Int. J. Artif. Intell. Tools, vol. 27, no. 7, Nov. 2018

[34] L. Genga, D. Potena, A. Chiorrini, C. Diamantini, and N. Zannone, "A Latitudinal Study on the Use of Sequential and Concurrency Patterns in Deviance Mining," Stud. Comput. Intell., vol. 880, pp. 103–119, 2020

[35] X. Zhao, J. Liang, and C. Dang, "A stratified sampling based clustering algorithm for large-scale data," Knowledge-Based Syst., vol. 163, pp. 416–428, Jan. 2019

[36] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," Sov. Phys. - Dokl., vol. 10, no. 8, pp. 707–710, 1966.

[37] P. Ceravolo, E. Damiani, M. Torabi, and S. Barbon, "Toward a new generation of log pre-processing methods for process mining," in Lecture Notes in Business Information Processing, 2017, vol. 297, pp. 55–70

[38] 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

[39] Z. Huang, W. Dong, F. Wang, and H. Duan, "Medical Inpatient Journey Modeling and Clustering: A Bayesian Hidden Markov Model Based Approach," AMIA ... Annu. Symp. proceedings. AMIA Symp., vol. 2015, pp. 649–658, 2015.

[40] G. M. Veiga and D. R. Ferreira, Understanding spaghetti models with sequence clustering for ProM, vol. 43 LNBIP. Springer Verlag, 2010, pp. 92–103.

[41] D. R. Ferreira, "Applied sequence clustering techniques for process mining," in Handbook of Research on Business Process Modeling, 2009, pp. 481–502.

[42] S. Kanj, T. Brüls, and S. Gazut, "Shared Nearest Neighbor Clustering in a Locality Sensitive Hashing Framework," J. Comput. Biol., vol. 25, no. 2, pp. 236–250, Feb. 2018

[43] H. Nguyen, M. Dumas, A. H. ter Hofstede, M. La Rosa, F. Maria Maggi, and M. Maggi, "Stage-based Discovery of Business Process Models from Event Logs."

[44] M. Boltenhagen, T. Chatain, and J. Carmona, Generalized Alignment-Based Trace Clustering of Process Behavior, vol. 11522 LNCS. Springer Verlag, 2019, pp. 237–257.

[45] A. Seeliger, T. Nolle, and M. Mühlhäuser, "Finding structure in the unstructured: Hybrid feature set clustering for process discovery," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 11080 LNCS, pp. 288–304, 2018

[46] M. La Rosa, M. Dumas, Chathura C Ekanayake, L. García-Bañuelos, J. Recker, and A. H. M. ter Hofstede, "Detecting approximate clones in business process model repositories," *Inf. Syst.*, vol. 49, pp. 102–125, 2015.

[47] P. Delias, M. Doumpos, E. Grigoroudis, and N. Matsatsinis, "A non-compensatory approach for trace clustering," *Int. Trans. Oper. Res.*, vol. 26, no. 5, pp. 1828–1846, Sep. 2019

Author Index