

4th International Symposium on

Data-Driven Process Discovery and Analysis SIMPDA 2014

*November 19-21, 2014
Milan, Italy*

*Editors:
Paolo Ceravolo
Rafael Accorsi
Barbara Russo*

Foreword

The fourth edition of the International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2014) conceived to offer a forum where researchers from different communities and the industry can share their insight in this hot new field.

With the increasing automation of business processes, growing amounts of process data become available. This opens new research opportunities for business process data analysis, mining and modeling. The aim of the IFIP 2.6 - International Symposium on Data-Driven Process Discovery and Analysis is to offer a forum where researchers from different communities and the industry can share their insight in this hot new field.

Submissions aim at covering theoretical issues related to process representation, discovery and analysis or provide practical and operational experiences in process discovery and analysis. Language for papers and presentations is English. In this fourth edition, 20 papers were submitted that were reviewed by a minimum of two reviewers, 13 was accepted for publication in the pre-symposium volume. According to the format of a symposium the discussion during the event is considered a valuable element that can help to improve a work presented and the approach in presenting results. For this reason authors of accepted papers will be invited to submit extended articles a post-symposium volume of Lecture Notes in Business Information Processing, scheduled in 2015.

Our thanks go to the authors who submitted to the conference, to the board of reviewers that made a great work in the review process and in promoting this new event, and to all those who participate in the organization of the events.

We are very grateful to Università degli Studi di Milano, for its financial support, IFIP, the University of Freiburg, the Free University of Bozen-Bolzano.

Paolo Ceravolo
Rafael Accorsi
Barbara Russo
SIMPDA co-Chairs

Table of Contents

* Research Papers

History-based Construction of Log-ProcessAlignments for Conformance Checking:Discovering What Really Went Wrong <i>M. Alizadeh, M. de Leoni and N. Zannone</i>	pp. 1-15
Finding Suitable Activity Clusters for Decomposed Process Discovery <i>B.F.A. Hompes, H.M.W. Verbeek and W.M.P van der Aalst</i>	pp. 16-30
Discovery of Frequent Episodes in Event Logs <i>M. Leemans and W. van der Aalst</i>	pp. 31-45
Business Process Measurement in Small Enterprises after the Installation of an ERP Software <i>S. Siccardi and C. Sebastiani</i>	pp. 46-59
Reasoning on Data-Aware Business Processes with Constraint Logic <i>M. Proietti and F. Smith</i>	pp. 60-75
CoPra2Go: an APP for Coding Collaboration Processes <i>E. Nowakowski, I. Seeber, R. Maier and F. Frati</i>	pp. 76-90
Scalable Dynamic Business Process Discovery with the Constructs Competition Miner <i>D. Redlich, T. Molka, W. Gilani, G. Blair and A. Rashid</i>	pp. 91-107
Scalable Process Monitoring through Rules and Neural Networks <i>A. Perotti, G. Boella and A. D'Avila Garcez</i>	pp. 108-122
Using Monotonicity to Find Optimal Process Configurations Faster <i>D. Schunselaar, E. Verbeek, H. A. Reijers and W. van der Aalst</i>	pp. 123-137

* Demonstrations

Tracking Hot Topics for the Monitoring of Open-World Processes

R. Pareschi, M. Rossetti and F. Stella

pp. 138-149

Using Semantic Lifting for Improving Educational Process Models
Discovery and Analysis

*A. Hicheur, J. Assu Ondo, B. Gueni, M. Fhima, M. Schwarcfeld
and N. Khelifa*

pp. 150-161

From Declarative Processes to Imperative Models

J. Prescher, C. Di Ciccio and J. Mendling

pp. 162-173

* Research Plans

A Methodology for Generating Artificial Event Logs to Compare
Process Discovery Techniques

T. Jouck, M. Swennen and B. Depaire

pp. 174-178

Process Mining Extension To SCAMPI

A. Valle

pp. 179-183

Conference Organization

* Conference Co-Chairs



Paolo Ceravolo

Università degli Studi di Milano,
Italy



Rafael Accorsi

University of Freiburg, Germany



Barbara Russo

Free University of Bozen-Bolzano

* Advisory Board

Karl Aberer, EPFL, Switzerland

Ernesto Damiani, Università degli Studi di Milano, Italy

Tharam Dillon, La Trobe University, Australia

Marcello Leida, EBTIC (Etisalat BT Innovation Centre), UAE

Erich Neuhold, University of Vienna, Austria

Maurice van Keulen, University of Twente, The Netherlands

Philippe Cudre-Mauroux, University Of Fribourg, Switzerland

* Ph.D. Award Committee

Gregorio Piccoli, Zucchetti spa, Italy

Paolo Ceravolo, Università degli Studi di Milano, Italy

Marcello Leida, EBTIC (Etisalat BT Innovation Centre), UAE

* Publicity Chair

Fulvio Frati, Università degli Studi di Milano, Italy

Program Committee

Irene Vanderfeesten, Eindhoven University Of Technology, The Netherlands

Maurice Van Keulen, University Of Twente, The Netherlands

Manfred Reichert, University Of Ulm, Germany

Schahram Dustdar, Vienna University Of Technology, Austria

Mohamed Mosbah, University Of Bordeaux, France

Meiko Jensen, Ruhr-University Bochum, Germany

Helen Balinsky, Hewlett-Packard Laboratories, Uk

Valentina Emilia Balas, University Of Arad, Romania

Karima Boudaoud, Ecole Polytechnique De Nice Sophia Antipolis, France

George Spanoudakis, City University London, Uk

Richard Chbeir, University Of Bourgogne, France

Gregorio Martinez Perez, University Of Murcia, Spain

Ebrahim Bagheri, Ryerson University, Canada

Jan Mendling, Vienna University Of Economics And Business, Austria

Farookh Hussain, University Of Technology Sydney, Australia

Marcello Leida, Ebtic (Etisalat Bt Innovation Centre), Uae

Wil Van Der Aalst, Technische Universiteit Eindhoven, The Netherlands

Ronald Maier, University Of Innsbruck, Austria

Chintan Amrit, University Of Twente, The Netherlands

Marco Montali, Free University Of Bozen - Bolzano, Italy

Elizabeth Chang, University New South Wales, Australia

Peter Spyns, Flemish Government, Belgium

Angelo Corallo, University Of Salento, Italy

Antonio Maña Gómez, University Of Málaga, Spain

Mustafa Jarrar, Birzeit University, Palestinian Territory

Isabella Seeber, University Of Innsbruck, Austria

Chi Hung, Tsinghua University, China

Alessandra Toninelli, Engineering Group, Italy
Haris Mouratidis, University Of Brighton, Uk
Abder Koukam, University Of Technology, Utm France
Fabrizio Maria Maggi, University Of Tartu, Estonia
Massimiliano De Leoni, Eindhoven Tu, Netherlands
Edgar Weippl, Tu Vienna, Austria
Pnina Soffer, University Of Haifa, Israel
Jianmin Wang, Tsinghua University Beijing, China
Minseok Song, Unist, South Korea
Roland Rieke, Fraunhofer Sit, Germany
Josep Carmona, Upc - Barcelona, Spain
Mark Strembeck, Wu Vienna, Austria
Matthias Weidlich, Imperial College, Uk
Mohamed Mosbah, University Of Bordeaux
Maria Leitner, University Of Vienna, Austria
Benoit Depaire, University Of Hasselt, Belgium
Barbara Weber, University Of Innsbruck, Austria
BabigaBirregah, University Of Technology Of Troyes, France

Sponsors



UNIVERSITÀ
DEGLI STUDI
DI MILANO



A.I.C.A.

Associazione Italiana per l'Informatica
e il Calcolo Automatico



FREIE UNIVERSITÄT BOZEN
LIBERA UNIVERSITÀ DI BOLZANO
FREE UNIVERSITY OF BOZEN • BOLZANO



ifip

History-based Construction of Log-Process Alignments for Conformance Checking: Discovering What Really Went Wrong^{*}

Mahdi Alizadeh, Massimiliano de Leoni, and Nicola Zannone

Department of Mathematics and Computer Science,
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
`{m.alizadeh,m.d.leoni,n.zannone}@tue.nl`

Abstract. Alignments provide a robust approach for conformance checking which has been largely applied in various contexts such as auditing and performance analysis. Alignment-based conformance checking techniques pinpoint the deviations causing nonconformity based on a cost function. However, such a cost function is often manually defined on the basis of human judgment and thus error-prone, leading to alignments that do not provide the most probable explanations of nonconformity. This paper proposes an approach to automatically define the cost function based on information extracted from the past process executions. The cost function only relies on objective factors and thus enables the construction of the most probable alignments, i.e. alignments that provide the most probable explanations of nonconformity. Our approach has been implemented in ProM and assessed using both synthetic and real-life data.

Keywords: Conformance checking, alignments, cost functions

1 Introduction

Modern organizations are centered around the processes needed to deliver products and services in an efficient and effective manner. Organizations that operate at a higher process maturity level use formal/semiformal models (e.g., UML, EPC, BPMN and YAWL models) to document their processes. In some case these models are used to configure process-aware information systems (e.g., WFM or BPM systems). However, in most organizations process models are not used to enforce a particular way of working. Instead, process models are used for discussion, performance analysis (e.g., simulation), certification, process improvement, etc. However, reality may deviate from such models. People tend to focus on idealized process models that have little to do with reality. This illustrates the importance of *conformance checking* [1,2,9].

^{*} This work has been funded by the NWO CyberSecurity programme under the PriCE project and the Dutch national program COMMIT under the THeCS project.

Conformance checking aims to verify whether the observed behavior recorded in an event log matches the intended behavior represented as a process model. The notion of alignments [2] provides a robust approach to conformance checking, which makes it possible to pinpoint the deviations causing nonconformity. An alignment between a recorded process execution and a process model is a pairwise matching between activities recorded in the log and activities allowed by the model. Sometimes, activities as recorded in the event log (events) cannot be matched to any of the activities allowed by the model (process activities). For instance, an activity is executed when not allowed. In this case, we match the event with a special *null* activity (hereafter, denoted as \gg), thus resulting in so-called *moves on log*. Other times, an activity should have been executed but is not observed in the event log. This results in a process activity that is matched to a \gg event, thus resulting in a so-called *move on model*.

Alignments are powerful artifacts to detect nonconformity between the observed behavior as recorded in the event log and the prescribed behavior as represented by process models. In fact, when an alignment between a log trace and process model contains at least one move on log or model, it means that such a log trace does not conform the model. As a matter of fact, the moves on log/model indicate where the execution is not conforming by pinpointing the deviations that have caused this nonconformity.

In general, a large number of possible alignments exist between a process model and a log trace, since there may exist manifold explanations why a trace is not conforming. It is clear that one is interested in finding the most probable explanation. Adriansyah et al. [4] have proposed an approach based on the principle of the Occam’s razor: the simplest and most parsimonious explanation is preferable. Therefore, one should not aim to find any alignment but, precisely, one of the alignments with the least expensive deviations (one of the so-called *optimal alignments*), according to some function assigning costs to deviations.

Existing alignment-based conformance checking techniques (e.g. [2,4]) require process analysts to manually define a cost function based on their background knowledge and beliefs. The definition of such a cost function is fully based on human judgment and, thus, prone to imperfections. These imperfections ultimately lead to alignments that are optimal, according to the provided cost function, but that do not provide the most probable explanation of nonconformity.

In this paper, we propose an alternative way to define a cost function, where the human judgment is put aside and only objective factors are considered. The cost function is automatically constructed by looking at the logging data and, more specifically, at the past process executions that are compliant with the process model. The intuition behind is that one should look at the past history of process executions and learn from it what is the most probable explanations of nonconformity. We believe that the most probable explanation of nonconformity of a certain process execution can be obtained by analyzing the behavior observed for such a process execution in each and every state and the behavior observed for other confirming traces when they were in the same state. Our approach

gives a potentially different cost for each move on model and log (depending on the current state), leading to the definition of a more sensitive cost function.

The approach has been fully implemented as a software plug-in for the open-source process-mining framework *ProM*. To assess the practical relevance of our approach, we performed an evaluation using both synthetic and real event logs and process models. In particular, we tested it on a real-life case study about the management of road-traffic fines by an Italian town. The results show that our approach significantly improves the accuracy in determining the most probable explanation for nonconformity compared to existing techniques.

The paper is organized as follows. Section 2 introduces preliminary concepts. Section 3 presents our approach for constructing optimal alignments. Section 4 presents experiment results, which are discussed in Section 5. Finally, Section 6 discusses related work and concludes the paper with directions for future work.

2 Preliminaries

This section introduces the notation and preliminaries for our work.

2.1 Labeled Petri Nets, Event Logs, and Alignments

Process models describe how processes should be carried out. Many languages exist to model processes. Here, we use a very simple formalism, which however allow one to define all the aspects to take into account for this paper:

Definition 1 (Labeled Petri Net). *A Labeled Petri net is a tuple $(P, T, F, A, \ell, m_i, m_f)$ where*

- P is a set of places;
- T is a set of transitions;
- $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation between places and transitions (and between transitions and places);
- A is the set of labels for transitions;
- $\ell : T \rightarrow A$ is a function that associates a label with every transition in T ;
- m_i is the initial marking;
- m_f is the final marking.

The label of a transition identifies the activity represented by such a transition. Multiple transitions can be associated with the same activity label; this means that the same activity is represented by multiple transitions. This is typically done to make the model simpler. Some transitions can be invisible. Invisible transitions do not correspond to actual activities but are necessary for routing purposes and, as such, their execution is never recorded in event logs. Given a Labeled Petri net N , $\text{Inv}_N \subseteq A$ indicates the set of labels associated with invisible transitions. As a matter of fact, invisible transitions are also associated with labels, though these labels do not represent activities. We assume that a label associated with a visible transition cannot be also associated with invisible ones and vice versa.

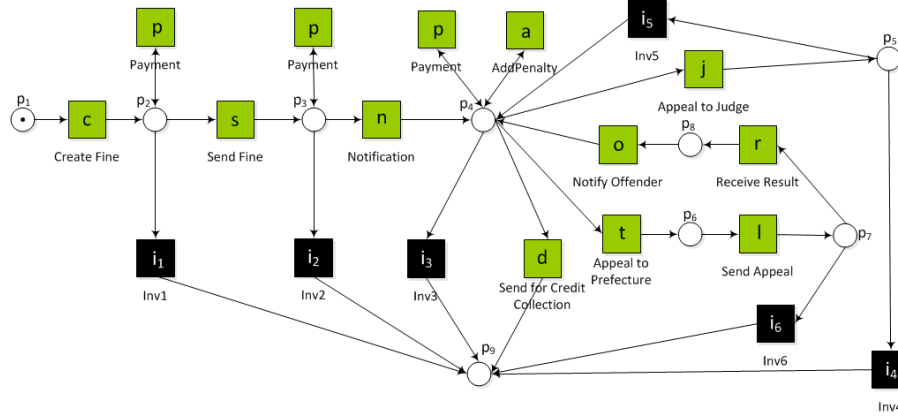


Fig. 1: A process model for managing road traffic fines. The green boxes represent the transitions that are associated with process activities while the black boxes represent invisible transitions. The text below the transitions represents the label, which is shortened with a single letter as indicated inside the transitions.

In the remainder, the simpler term Petri net is used to refer to Labeled Petri nets. The state of a Petri net is represented by a *marking*, i.e. a multiset of tokens on the places of the net. A Petri net has an initial marking m_i and a final marking m_f . When a transition is executed (i.e., *fired*), a token is taken from each of its input places and a token is added to each of its output places. A sequence of transitions σ_M leading from the initial to the final marking is a *complete process trace*. Given a Petri net $N = (P, T, F, A, \ell, m_i, m_f)$, T_N indicates the set of all complete process traces.

Example 1. Fig. 1 shows a process model for handling road traffic fines in Italy [14]. A process execution starts with recording a traffic fine in the system and sending it to Italian residents. Traffic fines might be paid before or after they are sent out by police or received by the offenders. If the fine is not paid in 180 days, a penalty is added. In addition, offenders may appeal against fines to prefecture and/or judge. If an appeal is accepted, the fine management is closed. Finally, if the fine is not paid by the offender, eventually the process terminates by handing over the case for credit collection.

Given a Petri net $N = (P, T, F, A, \ell, m_i, m_f)$, a log trace $\sigma_L \in A^*$ is a sequence of events where each event records the firing of a transition. In particular, each event records the label of the transition that has fired. An *event log* $\mathcal{L} \in \mathbb{B}(A)$ is a multiset of log traces, where $\mathbb{B}(X)$ is used to represent the set of all multisets over X . Here we assume that no events exist for activities not in A ; in practice, this can happen: in such cases, such events are filtered out before the event log is taken into consideration.

Not all log traces can be reproduced by a Petri net, i.e. not all log traces perfectly fit the process description. If a log trace perfectly fits the net, each

$$\gamma_1 = \begin{array}{|c|c|c|c|c|c|c|c|} \hline c & s & n & t & \gg & \gg & o & \gg \\ \hline c & s & n & t & l & r & o & i_3 \\ \hline \end{array} \quad \gamma_2 = \begin{array}{|c|c|c|c|c|c|c|c|} \hline c & s & n & t & o & \gg & \gg & \\ \hline c & s & n & t & \gg & l & i_6 & \\ \hline \end{array} \quad \gamma_3 = \begin{array}{|c|c|c|c|c|c|c|c|} \hline c & s & n & t & o & \gg & & \\ \hline c & s & n & \gg & \gg & & & d \\ \hline \end{array}$$

Fig. 2: Alignments of $\sigma_1 = \langle c, s, n, t, o \rangle$ and the process model in Fig. 1

“move” in the log trace, i.e. an event observed in the trace, can be mimicked by a “move” in the model, i.e. a transition fired in the net. After all events in the log trace are mimicked, the net reaches its final marking. In cases where deviations occur, some moves in the log trace cannot be mimicked by the net or vice versa. We explicitly denote “no move” by \gg .

Definition 2 (Legal move). Let $N = (P, T, F, A, \ell, m_i, m_f)$ be a Petri net. Let $S_L = (A \setminus \text{Inv}_N) \cup \{\gg\}$ and $S_M = A \cup \{\gg\}$. A legal move is a pair $(m_L, m_M) \in (S_L \times S_M) \setminus (\gg, \gg)$ such that

- (m_L, m_M) is a synchronous move if $m_L \in S_L$, $m_M \in S_M$ and $m_L = m_M$,
- (m_L, m_M) is a move on log if $m_L \in S_L$ and $m_M = \gg$,
- (m_L, m_M) is a move on model if $m_L = \gg$ and $m_M \in S_M$.

Σ_N denotes the set of legal moves for a Petri net N .

In the remainder, we indicate that a sequence σ' is a prefix of a sequence σ'' , denoted with $\sigma' \in \text{prefix}(\sigma'')$, if there exists a sequence σ''' such that $\sigma'' = \sigma' \oplus \sigma'''$, where \oplus denotes the concatenation operator.

Definition 3 (Alignment). Let Σ_N be the set of legal moves. An alignment of a log trace σ_L and a Petri net $N = (P, T, F, A, \ell, m_i, m_f)$ is a sequence $\gamma \in \Sigma_N^*$ such that, ignoring all occurrences of \gg , the projection on the first element yields σ_L and the projection on the second element yields a sequence $\langle a_1, \dots, a_n \rangle$ such that there exists a sequence $\sigma'_P = \langle t_1, \dots, t_n \rangle \in \text{prefix}(\sigma_P)$ for some $\sigma_P \in \Gamma_N$ where, for each $1 \leq i \leq n$, $\ell(t_i) = a_i$. If $\sigma'_P \in \Gamma_N$, γ is called a complete alignment of σ_L and N .

Fig. 2 shows three possible complete alignments of a log trace $\sigma_1 = \langle c, s, n, t, o \rangle$ and the net in Fig. 1. The top row of an alignment shows the sequence of events in the log trace, and the bottom row shows the sequence of activities in the net (both ignoring \gg). Hereafter, we denote $|_L$ the projection of an alignment over the log trace and $|_P$ the projection over the net.

As shown in Fig. 2, there can be multiple possible alignments for a given log trace and process model. The quality of an alignment is measured based on a provided cost function $K : \Sigma_N^* \rightarrow \mathbb{R}_0^+$, which assigns a cost to each alignment $\gamma \in \Sigma_N^*$. Typically, the cost of an alignment is defined as the sum of the costs of the individual moves in the alignment. An **optimal alignment** of a log trace and a process trace is one of the alignments with the lowest cost according to the provided cost function.

As an example, consider a cost function that assigns to any alignment a cost equal to the number of moves on log and model for visible transitions. If moves on model for invisible transitions i_k are ignored, γ_1 has two moves on model, γ_2

has one move on model and one move on log, and γ_3 has one move on model and two moves on log. Thus, according to the cost function, γ_1 and γ_2 are two optimal alignments of σ_1 and the process model in Fig. 1.

2.2 State Representation

At any point in time, a sequence of execution of activities leads to some state, and this state depends on which activities have been performed and in which order. Accordingly, any process execution can be mapped onto a state. As discussed in [3], a *state representation function* takes care of this mapping:

Definition 4 (State Representation). *Let $(P, T, F, A, \ell, m_i, m_f)$ be a Petri net. Let R be the set of possible state representations of sequences in A^* . A state representation function $\text{abst} : A^* \rightarrow R$ produces a state representation $\text{abst}(\sigma)$ for each process trace $\sigma \in \Gamma$.*

Several state-representation functions can be defined. Each function leads to a different abstraction, meaning that multiple different traces can be mapped onto the same state, thus abstracting out certain trace's characteristics. The following are examples of state-representation functions:

Sequence abstraction. It is a trivial mapping where the abstraction preserves the order of activities. Each trace is mapped onto a state that is the trace itself, i.e. for each $\sigma \in A^*$, $\text{abst}(\sigma) = \sigma$.

Multi-set abstraction. The abstraction preserves the number of times each activity is executed. This means that, for each $\sigma \in A^*$, $\text{abst}(\sigma) = M \in \mathbb{B}(A)$ such that, for each $a \in A$, M contains all instances of a in σ .

Set abstraction. The abstraction preserves whether each activity has been executed or not. This means that, for each $\sigma \in A^*$, $\text{abst}(\sigma) = M \subseteq A$ such that, for each $a \in A$, M contains a if it ever occurs in σ .

Example 2. Table 1 shows the state representation of some process traces of the net in Fig. 1 using different abstractions. For instance, trace $\langle c, p, p, s, n \rangle$ can be represented as the trace itself using the sequence abstraction, as $\{c(1), p(2), s(1), n(1)\}$ using the multi-set abstraction (in parenthesis the number of occurrences of activities in the trace), and as $\{c, p, s, n\}$ using the set abstraction. Traces $\langle c, p, s, n \rangle$ and $\langle c, p, p, s, n, p \rangle$ are also mapped to state $\{c, p, s, n\}$ using the set abstraction.

3 History-based Construction of the Most Probable Alignments

This section presents our approach to construct alignments that give the most probable explanations of deviations based on objective facts, i.e. the historical logging data, rather than on subjective cost functions manually defined by process analysts. To construct an optimal alignment between a process model and an event log, we use the A-star algorithm, analogously to what proposed in [4].

Section 3.1 discusses how the cost of an alignment is computed, whereas Section 3.2 briefly reports on the use of A-star to compute the most probable alignment.

Table 1: Examples of state representation using different abstractions

Sequence	#	Multi-set	#	Set	#
$\langle c, p \rangle$	25	$\{c(1), p(1)\}$	25	$\{c, p\}$	25
$\langle c, s, n, p \rangle$	15	$\{c(1), p(1), s(1), n(1)\}$	15		
$\langle c, p, p, s, n \rangle$	5	$\{c(1), p(2), s(1), n(1)\}$	5	$\{c, p, s, n\}$	45
$\langle c, p, p, s, n, p \rangle$	25	$\{c(1), p(3), s(1), n(1)\}$	25		
$\langle c, s, n, a, d \rangle$	10	$\{c(1), s(1), n(1), a(1), d(1)\}$	10	$\{c, s, n, a, d\}$	10
$\langle c, s, n, p, a, d \rangle$	10	$\{c(1), s(1), n(1), p(1), a(1), d(1)\}$	10	$\{c, s, n, p, a, d\}$	10
$\langle c, s, n, p, t, l \rangle$	25	$\{c(1), s(1), n(1), p(1), t(1), l(1)\}$	30		
$\langle c, s, p, n, t, l \rangle$	5			$\{c, s, n, p, t, l\}$	60
$\langle c, p, s, n, p, t, l \rangle$	5	$\{c(1), s(1), n(1), p(2), t(1), l(1)\}$	30		
$\langle c, s, p, n, p, t, l \rangle$	25				
$\langle c, s, n, p, t, l, r, o \rangle$	50	$\{c(1), s(1), n(1), p(1), t(1), l(1), r(1), o(1)\}$	50	$\{c, s, n, p, t, l, r, o\}$	50

3.1 Definition of cost functions

The computation of the most probable alignment relies on a cost function that accounts for the probability of an activity to be executed in a certain state. The definition of such a cost function requires an analysis of the past history as recorded in the event log to compute the probability of an activity to immediately occur or to never eventually occur when the process execution is in a certain state.

The cost of moves depends on probabilities. For this purpose, we need to introduce a functions' class $\mathcal{F} \subseteq [0, 1] \rightarrow \mathbb{R}^+$ such that $f \in \mathcal{F}$ if and only if $f(0) = \infty$ and f is monotonously decreasing between 0 and 1 (with $f(1) > 0$). Hereafter, these functions are called *cost profile*. It is easy to observe that if $f(p)$ is a cost-profile function, then $f(p)^i$ is also a cost-profile function for every $i > 0$. Examples of these functions are:

$$f(p) = \frac{1}{p} \quad f(p) = \frac{1}{\sqrt{p}} \quad f(p) = 1 + \log\left(\frac{1}{p}\right) \quad (1)$$

Similarly to what proposed in [4], the cost of an alignment move depends on the move type and the activity involved in the move but, differently from [4], it also depends on the position in which the move is inserted:

Definition 5 (Cost of an alignment move). Let $N = (P, T, F, A, \ell, m_i, m_f)$ be an Petri net. Let $\gamma \in \Sigma_N^*$ be a sequence of legal moves for N and $f \in \mathcal{F}$ a cost profile. The cost of appending a legal move $(m_L, m_M) \in \Sigma_N$ to γ with state-representation function *abst* is:

$$\kappa_{\text{abst}}((m_L, m_M), \gamma) = \begin{cases} 0 & m_L = m_M \\ 0 & m_L = \gg \text{ and } m_M \in \text{Inv}_N \\ f(P_{\text{abst}}(m_M \text{ occurs after } \gamma|_P)) & m_L = \gg \text{ and } m_M \notin \text{Inv}_N \\ f(P_{\text{abst}}(m_L \text{ never eventually occurs after } \gamma|_P)) & m_M = \gg \end{cases} \quad (2)$$

Readers can observe that the cost of a move on $\log(m_L, \gg)$ is not simply the probability of not executing activity m_L immediately after $\gamma|_P$; rather, it is the probability of never having activity m_M at the any moment in the future for that execution. This is motivated by the fact that a move on $\log(m_L, \gg)$ indicates that m_L is not expected to ever occur in the future. Conversely, if it was expected, a number of moves in model would be introduced until the process model, modeled as a Petri net, reaches a marking that allows m_L to occur (and, thus, a move in both can be appended). Different cost profiles account for the probabilities computed from historical logging data differently. In Section 4, we evaluate the cost profiles in Eq. 1 with different combinations of event logs and process models. The purpose is to verify whether a cost profile universally works better than the others. The following two definitions describe how to compute the probabilities required by Def. 5. For reliability, we only consider the subset of traces \mathcal{L}_{fit} of the original event log \mathcal{L} that comply with the process model.

Definition 6 (Probability that an activity occurs). *Let \mathcal{L} be an event log and $\mathcal{L}_{fit} \subseteq \mathcal{L}$ be the subset of traces that comply with a given process model represented by a Petri net $N = (P, T, F, A, \ell, m_i, m_f)$. The probability that an activity $a \in A$ occurs after executing σ with state-representation function abst is the ratio between number of traces in \mathcal{L}_{fit} in which a is executed after reaching state $\text{abst}(\sigma)$ and the total number of traces in \mathcal{L}_{fit} that reach state $\text{abst}(\sigma)$:*

$$P_{\text{abst}}(a \text{ occurs after } \sigma) = \frac{|\{\sigma' \in \mathcal{L}_{fit} : \exists \sigma'' \in \text{prefix}(\sigma'). \text{abst}(\sigma'') = \text{abst}(\sigma) \wedge \sigma'' \oplus \langle a \rangle \in \text{prefix}(\sigma')\}|}{|\{\sigma' \in \mathcal{L}_{fit} : \exists \sigma'' \in \text{prefix}(\sigma'). \text{abst}(\sigma'') = \text{abst}(\sigma)\}|} \quad (3)$$

Definition 7 (Probability that an activity never eventually occurs). *Let \mathcal{L} be an event log and $\mathcal{L}_{fit} \subseteq \mathcal{L}$ be the subset of traces that comply with a given process model represented by a Petri net $N = (P, T, F, A, \ell, m_i, m_f)$. The probability that an activity $a \in A$ will never eventually occur in a process execution after executing $\sigma \in A^*$ with state-representation function abst is the ratio between the number of traces in \mathcal{L}_{fit} in which a is never eventually executed after reaching state $\text{abst}(\sigma)$ and the total number of trace in \mathcal{L}_{fit} that reach state $\text{abst}(\sigma)$:*

$$P_{\text{abst}}(a \text{ never eventually occurs after } \sigma) = \frac{|\{\sigma' \in \mathcal{L}_{fit} : \exists \sigma'' \in \text{prefix}(\sigma'). \text{abst}(\sigma'') = \text{abst}(\sigma) \wedge \forall \sigma''' \sigma'' \oplus \sigma''' \oplus \langle a \rangle \in \text{prefix}(\sigma') \wedge a' \neq a\}|}{|\{\sigma' \in \mathcal{L}_{fit} : \exists \sigma'' \in \text{prefix}(\sigma'). \text{abst}(\sigma'') = \text{abst}(\sigma)\}|} \quad (4)$$

The cost of an alignment is the sum of the cost of all moves in the alignment, which are computed as described in Definition 5:

Definition 8 (Cost of an alignment). *The cost of alignment $\gamma \in \Sigma_N^*$ with state-representation function abst is computed as follows:*

$$K_{\text{abst}}(\gamma \oplus (m_L, m_M)) = \begin{cases} \kappa_{\text{abst}}((m_L, m_M), \langle \rangle) & \gamma = \langle \rangle \\ \kappa_{\text{abst}}((m_L, m_M), \gamma) + K_{\text{abst}}(\gamma) & \text{otherwise} \end{cases} \quad (5)$$

Hereafter, the term **most-probable alignment** is used to denote any of the optimal alignments (i.e., with the lowest cost) according to the cost function given in Definition 8.

3.2 The use of the A-star algorithm to construct alignments

The A-star algorithm [10] aims to find a path in a graph V from a given *source* node v_0 to any node $v \in V$ in a target set. Every node v of graph V is associated with a cost determined by an *evaluation* function $f(v) = g(v) + h(v)$, where

- $g : V \rightarrow \mathbb{R}_0^+$ is a function that returns the smallest path cost from v_0 to v ;
- $h : V \rightarrow \mathbb{R}_0^+$ is an heuristic function that estimates the path cost from v to its preferred target node.

Function h is said to be *admissible* if it returns a value that underestimates the distance of a path from a node v' to its preferred target node v'' , i.e. $g(v') + h(v') \leq g(v'')$. If h is admissible, A-star finds a path that is guaranteed to have the overall lowest cost.

The A-star algorithm keeps a priority queue of nodes to be visited: higher priority is given to nodes with lower costs. The algorithm works iteratively: at each step, the node v with lowest cost is taken from the priority queue. If v belongs to the target set, the algorithm ends returning node v . Otherwise, v is expanded: every successors v' is added to priority queue with a cost $f(v')$.

We employ A-star to find any of the optimal alignments between a log trace $\sigma_L \in \mathcal{L}$ and a Petri net N . In order to be able to apply A-star, an opportune search space needs to be defined. Every node γ of the search space V is associated to a different alignment that is a prefix of some complete alignment of σ_L and N . Since a different alignment is also associated to every search-space node and vice versa, we use the alignment to refer to the associated state. The source node is an empty alignment $\gamma_0 = \langle \rangle$ and the set of target nodes includes every complete alignment of σ_L and N .

Let us denote the length of a sequence σ with $\|\sigma\|$. Given a node/alignment $\gamma \in V$, the search-space successors of γ include all alignments $\gamma' \in V$ obtained from γ by concatenating exactly one move. Given an alignment $\gamma \in V$, the cost of path from the initial node to node $\gamma \in V$ is:

$$g(\gamma) = \|\gamma|_L\| + K(\gamma).$$

where $K(\gamma)$ is the cost of alignment γ according to Definition 8. It is easy to check that, given two complete alignments γ'_C and γ''_C , $K(\gamma'_C) < K(\gamma''_C)$ iff $g(\gamma'_C) < g(\gamma''_C)$ and $K(\gamma'_C) = K(\gamma''_C)$ iff $g(\gamma'_C) = g(\gamma''_C)$. Therefore, an optimal solution returned by A-star coincides with an optimal alignment. To define a more efficient and admissible heuristics, we consider term $\|\sigma_L\|$ in h ; this term does not affect optimality. Given an alignment $\gamma \in V$, we employ the heuristics:

$$h(\gamma) = \|\sigma_L\| - \|\gamma|_L\|.$$

For alignment γ , the number of steps to add in order to reach a complete alignment is lower bounded by the number of execution steps of trace σ_L that have not been included yet in the alignment, i.e. $\|\sigma_L\| - \|\gamma|_L\|$. Since the additional cost to traverse a single node is at least 1, the cost to reach a target node is at least $h(\gamma)$, corresponding to the case where the part of the log trace that still needs to be included in the alignment perfectly fits.

$$\gamma' = \underbrace{\begin{array}{|c|c|c|} \hline c & s & n \\ \hline c & s & n \\ \hline \end{array}}_{\gamma} \oplus \begin{cases} (l, \gg) & \kappa((l, \gg), \gamma) = 1.49 \\ (\gg, p) & \kappa((\gg, p), \gamma) = 1.04 \\ (\gg, a) & \kappa((\gg, a), \gamma) = 2.04 \\ (\gg, d) & \kappa((\gg, d), \gamma) = \infty \\ \dots & \end{cases}$$

Fig. 3: Construction of the alignment of log trace $\sigma_2 = \langle c, s, n, l, o \rangle$ and the net in Fig. 1. Cost of moves are computed with sequence state-representation function, cost profile $f(p) = 1 + \log(1/p)$, and \mathcal{L}_{fit} in Table 1.

Example 3. Consider a log trace $\sigma_2 = \langle c, s, n, l, o \rangle$ and the net N in Fig. 1. An analyst wants to determine the most probable explanations for nonconformity by constructing the most probable alignment of σ_2 and N , based on historical logging data. In particular, \mathcal{L}_{fit} consists of the traces in Table 1 (the first column shows the traces, and the second the number of occurrences of a trace in the history). Assume that the algorithm has constructed an optimal alignment γ of trace $\langle c, s, n \rangle \in \text{prefix}(\sigma_2)$ and N (left part of Fig. 3). The next event in the log trace (i.e., l) cannot be replayed in the net. Therefore, the algorithm should determine which move is the most likely to have occurred. Different moves are possible; for instance, a move on log for l , a move on model for p , a move on model for t , etc. The algorithm computes the cost for these moves using Eq. 5 (right part of Fig. 3). As move on model (\gg, p) is the move with the least cost (and no other alignments have lower cost), alignment $\gamma' = \gamma \oplus (\gg, p)$ is selected for the next iteration. It is worth noting that activity d never occurs after $\langle c, s, n \rangle$ in \mathcal{L}_{fit} ; consequently, the cost of move (\gg, d) is equal to ∞ .

4 Implementation and Experiments

We have implemented our approach for history-based construction of alignments as a plugin of the open-source ProM framework (<http://www.promtools.org>). The plug-in takes as inputs a process model and two event logs. It computes the most probable alignments for each trace in the first event log based on the frequency of the traces in the second event log (historical logging data).

To assess the practical feasibility and accuracy of the approach, we performed a number of experiments using both synthetic and real-life logs. In the experiments with synthetic logs, we assumed that the execution of an activity depends on the activities that were performed in the past. In the experiments with real-life logs, we tested if this assumption holds in real applications. Accordingly, the real-life logs were used as historical logging data. To evaluate the approach, we artificially added noise to the traces used for the experiments. This was necessary to assess the ability of the approach to reconstruct the original traces.

4.1 Synthetic Data

For the experiments with synthetic data, we used the process for handling credit requests in [14]. Based on this model, we generated 10000 traces consisting of

Table 2: Results of experiments on synthetic data. CA indicates the percentage of correct alignments, and LD indicates the overall Levenshtein distance between the original traces and the projection of the alignments over the process. For comparison with existing approaches, the standard cost function as defined in [4] was used. In bold the best result is highlighted for each amount of noise.

	1/p						1/√p						1 + log(1/p)						Existing approach	
	Seq		Multi-set		Set		Seq		Multi-set		Set		Seq		Multi-set		Set			
Noise	CA	LD	CA	LD	CA	LD	CA	LD	CA	LD	CA	LD	CA	LD	CA	LD	CA	LD	CA	LD
10%	93	259	93	258	87	514	95	164	95	164	88	430	95	153	95	154	88	409	92	233
20%	85	569	85	561	78	968	87	426	87	431	79	852	87	410	87	415	79	823	83	534
30%	74	1084	74	1077	65	1653	76	950	75	963	66	1509	76	944	75	958	67	1474	71	1110
40%	63	1658	62	1659	55	2285	64	1519	64	1537	56	2148	64	1512	64	1535	56	2118	60	1685

69504 events using the CPN Tools (<http://cpntools.org>). To assess the accuracy of the approach, we manipulated 20% of these traces by introducing different percentages of noise. In particular, given a trace, we added and removed a number of activities to/from the trace equal to the same percentage of the trace length. The other traces were used as historical logging data. We computed the most probable alignments of the manipulated traces and process model, and evaluated the ability of the approach to reconstruct the original traces. To this end, we measured the percentage of correct alignments (i.e., the cases where a projection of an alignment over the process coincides with the original trace) and compute the overall Levenshtein distance [12] between the original traces and the projection of the computed alignments over the process. This string metric measures the distance between two sequences, i.e. the minimal number of changes required to transform one sequence into the other. In our setting, it provides an indication of how much the projection of the computed alignments over the process is close to the original traces.

We tested our approach with different amounts of noise (i.e., 10%, 20%, 30% and 40% of the trace length), with different cost profiles (i.e., $1/p$, $1/\sqrt{p}$, and $1 + \log(1/p)$), and with different state-representation functions (i.e., *sequence*, *multi-set*, and *set*). Moreover, we compared our approach with existing alignment-based conformance checking techniques. In particular, we used the standard cost function introduced in [4]. We repeated each experiment five times. Table 2 shows the results where every entry reports the average over the five runs.

The results show that cost profiles $1/\sqrt{p}$ and $1 + \log(1/p)$ in combination with *sequence* and *multi-set* abstractions are able to better identify what really happened, i.e. they align the manipulated traces with the corresponding original traces in more cases (CA). In all cases, cost profile $1 + \log(1/p)$ with *sequence* state-representation function provides more accurate diagnostics (LD): even if log traces are not aligned to the original traces, the projection over the process of alignments constructed using this cost profile and abstraction are closer to the original traces. Compared to the cost function used in [4], our approach computed the correct alignment for 4.4% more traces when cost profile $1 + \log(1/p)$ and *sequence* state-representation function are used. In particular, our approach correctly reconstructed the original trace for 18.4% of the traces that were not

Table 3: Results of experiments on real-life data. Notation analogous to Table 2.

	1/p						1/√p						1 + log(1/p)						Existing approach	
	Seq		Multi-set		Set		Seq		Multi-set		Set		Seq		Multi-set		Set			
Noise	CA	LD	CA	LD	CA	LD	CA	LD	CA	LD	CA	LD	CA	LD	CA	LD	CA	LD	CA	LD
10%	99	397	99	397	99	415	99	384	99	389	99	408	99	366	99	371	99	389	98	1274
20%	99	585	99	585	99	602	99	570	99	575	99	592	99	554	99	559	99	576	97	1448
30%	89	3349	89	3349	89	3371	89	3300	89	3341	89	3362	89	3281	89	3322	89	3344	87	4284
40%	76	9160	76	9160	75	9238	76	9091	76	9152	75	9230	76	9103	75	9165	75	9243	74	9861

correctly reconstructed using the cost function used in [4]. Moreover, an analysis of LD shows that, on average, the traces reconstructed using our approach have 0.37 deviations, while the traces reconstructed using the cost function used in [4] have 0.45 deviation. This corresponds to an improvement of LD of about 15.2%.

4.2 Real-life Logs

To evaluate the applicability of our approach to real-life scenarios, we used an event log obtained from a fine management system of the Italian police [14]. The process model in form of Petri net is presented in Fig. 1. We extracted a log consisting of 142408 traces and 527549 events, where all traces are conforming with the net. To these traces, we applied the same methodology used for the experiments reported in Section 4.1. We repeated the experiments five times. Table 3 shows the results where every entry reports the average over five runs.

The results confirm that cost profiles $1/\sqrt{p}$ and $1 + \log(1/p)$ in combination with *sequence* and *multi-set* state-representation functions provide the more accurate diagnostics (both CA and LD). Moreover, the results show that our approach (regardless of the used cost profile and state-representation function) performs better than the cost function in [4] on real-life logs. In particular, using *sequence* state-representation function and cost profile $1 + \log(1/p)$, our approaches computed the correct alignment for 1.8% more traces than what the cost function in [4] did. In particular, our approach correctly reconstructed the original trace for 19.3% of the traces that were not correctly reconstructed using the cost function used in [4]. Moreover, our approach improves LD by 21.1% compared to the cost function used in [4]. Such an improvement shows that when the original trace is not reconstructed correctly, our approach returns an explanation that is significantly closer to the actual explanation.

5 Discussion

The A-star algorithm requires a cost function to penalize nonconformity. In our experiments, we have considered a number of cost profiles to compute the cost of moves on log/model based on the probability of a given activity to occur in historical logging data. The selection of the cost profile has a significant impact on the results as they penalize deviations differently. For instance, cost profile $1/p$ penalizes less probable moves much more than $1 + \log(1/p)$. To illustrate

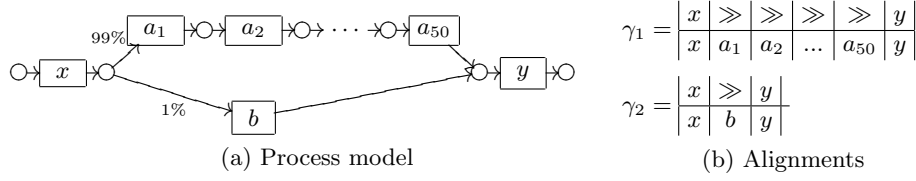


Fig. 4: Process model including two paths formed by a (sub)sequence of 50 activities and 1 activity respectively. The first path is executed in 99% of the cases; the second in 1% of the cases. γ_1 and γ_2 are two possible alignments of trace $\sigma = \langle x, y \rangle$ and the process model.

this, consider a trace $\sigma = \langle x, y \rangle$ and the process model in Fig. 4a. Two possible alignments, namely γ_1 and γ_2 , are conceivable (Fig. 4b). γ_1 contains a large number of deviations compared to γ_2 (50 moves on log vs. 1 move on log). The use of cost profile $1/p$ yields γ_1 as the most probable alignment, while the use of cost profile $1 + \log(1/p)$ yields γ_2 as the most probable alignment. Tables 2 and 3 show that cost profile $1 + \log(1/p)$ usually provides more accurate results. Cost profile $1/p$ penalizes less probable moves excessively, and thus tends to construct alignments with more frequent traces in the historical logging data even if those alignments contain a significantly larger number of deviations. Our experiments suggest that the construction of the most probable alignments requires a trade-off between the frequency of the traces in historical logging data and the number of deviations in alignments, which is better captured by cost profile $1 + \log(1/p)$.

Different state-representation functions can be used to characterize the state of a process execution. In this work, we have considered three state-representation functions: *sequence*, *multi-set*, and *set*. The experiments show that in general the sequence abstraction produces more accurate results compared to the other abstractions. The set abstraction provides the least accurate results, especially when applied to the process for handling credit requests (Table 2). The main reason is that this abstraction is not able to accurately characterize the state, especially in presence of loops: after each loop iteration the process execution yields the same state. Therefore, the cost function constructed using the set abstraction is not able to account for the fact that the probability of executing certain activities can increase after every loop iteration, thus leading to alignments in which loops are not captured properly.

To conclude, the experiments show that our technique tends to build alignments that give better explanations of deviations. It is easy to see that, when nonconformity is injected in fitting traces and alignments are subsequently built, the resulting alignments yield perfect explanations if the respective process projections coincide with the respective fitting traces before the injections of nonconformity. Tables 2 and 3 have shown that, basing the construction of the cost function on the analysis of historical logging data our technique tends to build alignments whose process projection is closer to the original fitting traces and, hence, the explanations of deviations are closer to the correct ones.

6 Related Work and Conclusions

In process mining, a number of approaches have been proposed to check conformance of process models and the actual behavior recorded in event logs. Some approaches [7,8,13,15,16] check conformance by verifying whether traces satisfies rules encoding properties expected from the process. Petković et al. [17] verify whether a log trace is a valid trace of the transition system generated by the process model. Token-based approaches [6,18] use the number of missing and added tokens obtained by replaying traces over the process model to measure the conformance between the log and the process. However, these approaches only give a boolean answers diagnosing whether traces conform to a process model or not. When they are able to provide diagnostic information, such information is often imprecise. For instance, token-based approaches may allow behavior that is not allowed by the model due to the used heuristics and thus may provide incorrect diagnostic information.

Recently, the construction of alignments has been proposed as a robust approach for checking the conformance of event logs with a given process model [4]. Alignments have proven to be very powerful artifacts to perform conformance checking. By constructing alignments, analysts can be provided with richer and more accurate diagnostic information. In fact, alignments are also used as the main enablers for a number of techniques for process analytics, auditing, and process improvement, such as for performance analysis [2], privacy compliance [5] and process-model repairing [11].

To our knowledge, the main problem of existing techniques for constructing optimal alignments is related to the fact that process analysts need to provide a function which associates a cost to every possible deviation. These cost functions are only based on human judgment and, hence, prone to imperfections. If these techniques are fed with imprecise cost functions, they create imperfect alignments, which ultimately leads to unlikely or, even, incorrect diagnostics.

In this paper, we have proposed a different approach where the cost function is automatically computed based on real facts: historical logging data recorded in event logs. In particular, the cost function is computed based on the probability of activities to be executed or not in a certain state (representing which activities have been executed and their order). Experiments have shown that, indeed, our approach can provide more probable explanations of nonconformity of process executions, if compared with existing techniques.

We acknowledge that the evaluation is far from being completed. We aim to perform more extensive experiments to verify whether certain cost-profile functions provide more probable alignments than others or, at least, to give some guidelines to determine in which settings a given cost-profile function is preferable. We also aim to develop a technique that, given a model and log, allow for (semi-)automatic tuning of the cost-profile function and state abstraction.

In this paper, we only considered the control-flow, i.e. the name of the activities and their ordering, to construct the cost function and, hence, to compute the most probable alignment. However, the choice in a process execution is often driven by other aspects. For instance, when instances are running late, the

execution of certain fast activities are more probable; or, if a certain process attribute takes on a given value, certain activities are more likely to be executed. We expect that our approach can be significantly improved if the other business process perspectives (i.e., data, time and resources) are taken into account.

References

1. van der Aalst, W.M.P.: Process Mining - Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying history on process models for conformance checking and performance analysis. *Data Min. Knowl. Discov.* 2(2), 182–192 (2012)
3. van der Aalst, W.M.P., Schonenberg, M.H., Song, M.: Time prediction based on process mining. *Information Systems* 36(2), 450–475 (2011)
4. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Memory-efficient alignment of observed and modeled behavior. *BPM Center Report 03-03*, BPMcenter.org (2013)
5. Adriansyah, A., van Dongen, B.F., Zannone, N.: Privacy analysis of user behavior using alignments. *it-Information Technology* 55(6), 255–260 (2013)
6. Banescu, S., Petkovic, M., Zannone, N.: Measuring privacy compliance using fitness metrics. In: *Business Process Management*. pp. 114–119. LNCS 7481, Springer (2012)
7. Borrego, D., Barba, I.: Conformance checking and diagnosis for declarative business process models in data-aware scenarios. *Expert Syst. Appl.* 41(11), 5340–5352 (2014)
8. Caron, F., Vanthienen, J., Baesens, B.: Comprehensive rule-based compliance checking and risk management with process mining. *Decision Support Systems* 54(3), 1357–1369 (2013)
9. Cook, J.E., Wolf, A.L.: Software process validation: quantitatively measuring the correspondence of a process to a model. *TOSEM* 8(2), 147–176 (1999)
10. Dechter, R., Pearl, J.: Generalized best-first search strategies and the optimality of A*. *Journal of the ACM* 32, 505–536 (1985)
11. Fahland, D., van der Aalst, W.M.P.: Model repair - aligning process models to reality. *Information Systems* (2014)
12. Levenshtein, V.: Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady* 10(8), 707–710 (1966)
13. Ly, L.T., Rinderle-Ma, S., Göser, K., Dadam, P.: On enabling integrated process compliance with semantic constraints in process management systems - requirements, challenges, solutions. *Information Systems Frontiers* 14(2), 195–219 (2012)
14. Mannhardt, F., de Leoni, M., van der Aalst, W.M.P.: Balanced multi-perspective checking of process conformance. *BPM Center Report 14-08*, BPMcenter.org (2014)
15. de Medeiros, A.K.A., van der Aalst, W.M.P., Pedrinaci, C.: Semantic Process Mining Tools: Core Building Blocks. In: *Proc. ECIS*. pp. 1953–1964. AIS (2008)
16. Montali, M.: Specification and Verification of Declarative Open Interaction Models - A Logic-Based Approach. LNBIP 56, Springer (2010)
17. Petković, M., Prandi, D., Zannone, N.: Purpose control: Did you process the data for the intended purpose? In: *Secure Data Management*. pp. 145–168. LNCS 6933, Springer (2011)
18. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. *Information Systems* 33(1), 64–95 (2008)

Finding Suitable Activity Clusters for Decomposed Process Discovery

B.F.A. Hompes, H.M.W. Verbeek, and W.M.P. van der Aalst

Department of Mathematics and Computer Science
Eindhoven University of Technology, Eindhoven, The Netherlands
`b.f.a.hompes@student.tue.nl`
`{h.m.w.verbeek,w.m.p.v.d.aalst}@tue.nl`

Abstract. Event data can be found in any information system and provide the starting point for a range of process mining techniques. The widespread availability of large amounts of event data also creates new challenges. Existing process mining techniques are often unable to handle “big event data” adequately. *Decomposed process mining* aims to solve this problem by decomposing the process mining problem into many smaller problems which can be solved in less time, using less resources, or even in parallel. Many decomposed process mining techniques have been proposed in literature. Analysis shows that even though the decomposition step takes a relatively small amount of time, it is of key importance in finding a high-quality process model and for the computation time required to discover the individual parts. Currently there is no way to assess the quality of a decomposition beforehand. We define three quality notions that can be used to assess a decomposition, before using it to discover a model or check conformance with. We then propose a decomposition approach that uses these notions and is able to find a high-quality decomposition in little time.

Keywords: decomposed process mining, decomposed process discovery, distributed computing, event log

1 Introduction

Process mining aims to discover, monitor and improve real processes by extracting knowledge from event logs readily available in today’s information systems [1]. In recent years, (business) processes have seen an explosive rise in supporting infrastructure, information systems and recorded information, as illustrated by the term Big Data. As a result, event logs generated by these information systems grow bigger and bigger as more event (meta-)data is being recorded and processes grow in complexity. This poses both opportunities and challenges for the process mining field, as more knowledge can be extracted from the recorded data, increasing the practical relevance and potential economic value of process mining. Traditional process mining approaches however have difficulties coping with this sheer amount of data (i.e. the number of events), as most interesting

algorithms are linear in the size of the event log and exponential in the number of different activities [3]. In order to provide a solution to this problem, techniques for *decomposed process mining* [3–5] have been proposed. Decomposed process mining aims to decompose the process mining problem at hand into smaller problems that can be handled by existing process discovery and conformance checking techniques. The results for these individual sub-problems can then be combined into solutions for the original problems. Also, these smaller problems can be solved concurrently with the use of parallel computing. Even sequentially solving many smaller problems can be faster than solving one big problem, due to the exponential nature of many process mining algorithms. Several decomposed process mining techniques have been developed in recent years [2–5, 7, 8, 10, 12, 13]. Though existing approaches have their merits, they lack in generality. In [5], a generic approach to decomposed process mining is proposed. The proposed approach provides a framework which can be combined with different existing process discovery and conformance checking techniques. Moreover, different decompositions can be used while still providing formal guarantees, e.g. the fraction of perfectly fitting traces is not influenced by the decomposition. When decomposing an event log for (decomposed) process mining, several problems arise. In terms of decomposed process discovery, these problems lie in the step where the overall event log is decomposed into sublogs, where submodels are discovered from these sublogs, and/or where submodels are merged to form the final model. Even though creating a decomposition is computationally undemanding, it is of key importance for the remainder of the decomposed process discovery process in terms of the overall required processing time and the quality of the resulting process model.

The problem is that there is currently no clear way of determining the quality of a given decomposition of the events in an event log, before using that decomposition to either discover a process model or check conformance with. The current decomposition approaches do not use any quality notions to create a decomposition. Thus, potential improvements lie in finding such quality notions and a decomposition approach that uses those notions to create a decomposition with.

The remainder of this paper is organized as follows. In [Section 2](#) related work is discussed briefly. [Section 3](#) introduces necessary preliminary definitions for decomposed process mining and the generic decomposition approach. [Section 4](#) introduces decomposition quality notions to grade a decomposition upon, and two approaches that create a high quality decomposition according to those notions. [Section 5](#) shows a (small) use case. The paper is concluded with views on future work in [Section 6](#).

2 Related Work

Little work has been done on the decomposition and distribution of process mining problems [3–5]. In [14] MapReduce is used to scale event correlation as a preprocessing step for process mining. In [6] an approach is described to distribute genetic process mining over multiple computers. In this approach can-

didate models are distributed and in a similar fashion the log can be distributed as well. However, individual models are not partitioned over multiple nodes. More related are the divide-and-conquer techniques presented in [9], where it is shown that region-based synthesis can be done at the level of synchronized State Machine Components (SMCs). Also a heuristic is given to partition the causal dependency graph into overlapping sets of events that are used to construct sets of SMCs. In [4] a different (more local) partitioning of the problem is given which, unlike [9], decouples the decomposition approach from the actual conformance checking and process discovery approaches. The approach presented in this paper is an extension of the approach presented in [4]. Where [4] splits the process mining problem at hand into subproblems using a *maximal* decomposition, our approach first aims to recombine the many created activity clusters into better and fewer clusters, and only then splits the process mining problem into subproblems. As a result, fewer subproblems remain to be solved. The techniques used to recombine clusters are inspired by existing software quality metrics and the business process metrics listed in [15]. More information on the use of software engineering metrics in a process mining context is described there as well.

3 Preliminaries

This section introduces the notations needed to define a better decomposition approach. A basic understanding of process mining is assumed [1].

3.1 Multisets, Functions, and Sequences

Definition 1 (Multisets).

Multisets are defined as sets where elements may appear multiple times. $\mathcal{B}(A)$ is the set of all multisets over some set A . For some multiset $b \in \mathcal{B}(A)$, and element $a \in A$, $b(a)$ denotes the number of times a appears in b .

For example, take $A = \{a, b, c, d\}$: $b_1 = []$ denotes the empty multiset, $b_2 = [a, b]$ denotes the multiset over A where $b_2(c) = b_2(d) = 0$ and $b_2(a) = b_2(b) = 1$, $b_3 = [a, b, c, d]$ denotes the multiset over A where $b_3(a) = b_3(b) = b_3(c) = b_3(d) = 1$, $b_4 = [a, b, b, d, a, c]$ denotes the multiset over A where $b_4(a) = b_4(b) = 2$ and $b_4(c) = b_4(d) = 1$, and $b_5 = [a^2, b^2, c, d] = b_4$. The standard set operators can be extended to multisets, e.g. $a \in b_2$, $b_5 \setminus b_2 = b_3$, $b_2 \uplus b_3 = b_4 = b_5$, $|b_5| = 6$

Definition 2 (Sequences).

A sequence is defined as an ordering of elements of some set. Sequences are used to represent paths in a graph and traces in an event log. $\mathcal{S}(A)$ is the set of all sequences over some set A . $s = \langle a_1, a_2, \dots, a_n \rangle \in \mathcal{S}(A)$ denotes a sequence s over A of length n . Furthermore: $s_1 = \langle \rangle$ is the empty sequence and $s_1 \cdot s_2$ is the concatenation of two sequences.

For example, take $A = \{a, b, c, d\}$: $s_1 = \langle a, b, b \rangle$, $s_2 = \langle b, b, c, d \rangle$, $s_1 \cdot s_2 = \langle a, b, b, b, b, c, d \rangle$

Definition 3 (Function Projection).

Let $f \in X \not\rightarrow Y$ be a (partial) function and $Q \subseteq X$. $f|_Q$ denotes the projection of f on Q : $\text{dom}(f|_Q) = \text{dom}(f) \cap Q$ and $f|_Q(x) = f(x)$ for $x \in \text{dom}(f|_Q)$.

The projection can be used for multisets. For example, $b_5|_{\{a,b\}} = [a^2, b^2]$.

Definition 4 (Sequence Projection).

Let A be a set and $Q \subseteq A$ a subset. $\downarrow_Q \in \mathcal{S}(A) \rightarrow \mathcal{S}(Q)$ is a projection function and is defined recursively: (1) $\langle \rangle \downarrow_Q = \langle \rangle$ and (2) for $s \in \mathcal{S}(A)$ and $a \in A$:

$$(\langle a \rangle \cdot s) \downarrow_Q = \begin{cases} s \downarrow_Q & \text{if } a \notin Q \\ \langle a \rangle \cdot s \downarrow_Q & \text{if } a \in Q \end{cases}$$

So $\langle a, a, b, b, c, d, d \rangle \downarrow_{\{a,b\}} = \langle a, a, b, b \rangle$.

3.2 Event Logs

Event logs are the starting point for process mining. They contain information recorded by the information systems and resources supporting a process. Typically, the executed *activities* of multiple *cases* of a *process* are recorded. Note that only *example behavior* is recorded, i.e. event logs only contain information that has been seen. An event log often contains only a fraction of the possible behavior [1]. A trace describes one specific instance (i.e. one “run”) of the process at hand, in terms of the executed activities. An event log is a multiset of traces, since there can be multiple cases having the same trace. For the remainder of this paper, we let \mathcal{U}_A be some universe of activities.

Definition 5 (Trace).

Let $A \subseteq \mathcal{U}_A$ be a set of activities. A trace $s \in \mathcal{S}(A)$ is a sequence of activities.

Definition 6 (Event log).

Let $A \subseteq \mathcal{U}_A$ be a set of activities. Let $L \in \mathcal{B}(\mathcal{S}(A))$ be a multiset of traces over A . L is an event log over A .

An example event log is $L_1 = [\langle a, b, c, d \rangle^5, \langle a, b, b, c, d \rangle^2, \langle a, c, d \rangle^3]$. There are three unique traces in L_1 , and it contains information about a total of 10 cases. There are $4 \cdot 5 + 5 \cdot 2 + 3 \cdot 3 = 39$ events in total. The projection can be used for event logs as well. That is, for some log $L \in \mathcal{B}(\mathcal{S}(A))$ and set $Q \subseteq A$: $L|_Q = [s|_Q | s \in L]$. For example $L_1|_{\{a,b,c\}} = [\langle a, b, c \rangle^5, \langle a, b, b, c \rangle^2, \langle a, c \rangle^3]$. We will refer to these projected event logs as *sublogs*.

3.3 Activity Matrices, Graphs, and Clusters

In [5] different steps for a generic decomposed process mining approach have been outlined. In [16], an implementation of the generic approach has been created which *decomposes* the overall event log based on a *causal graph* of activities. This section describes the necessary definitions for this decomposition method.

Definition 7 (Causal Activity Matrix).

Let $A \subseteq \mathcal{U}_A$ be a set of activities. $\mathcal{M}(A) = (A \times A) \rightarrow [-1.0, 1.0]$ denotes the set of causal activity matrices over A . For $a, a' \in A$ and $M \in \mathcal{M}(A)$, $M(a, a')$ denotes the “directly follows strength” from a to a' .

A $M(a, a')$ value close to 1.0 signifies that we are quite confident there exists a directly follows relation between two activities while a value close to -1.0 signifies that we are quite sure there is no relation. A value close to 0.0 indicates uncertainty, i.e., there may be a relation, but there is no strong evidence for it.

Definition 8 (Causal Activity Graph).

Let $A \subseteq \mathcal{U}_A$ be a set of activities. $\mathcal{G}(A)$ denotes the set of causal activity graphs over A . A causal activity graph $G \in \mathcal{G}(A)$ is a 2-tuple $G = (V, E)$ where $V \subseteq A$ is the set of nodes and $E \subseteq (V \times V)$ is the set of edges. $G = (V, E) \in \mathcal{G}(A)$ is the causal activity graph based on $M \in \mathcal{M}(A)$ and a specific causality threshold $\tau \in [-1.0, 1.0]$ iff $E = \{(a, a') \in A \times A \mid M(a, a') > \tau\}$ and $V = \bigcup_{(a, a') \in E} \{a, a'\}$. That is, for every pair of activities $(a, a') \in A$, there’s an edge from a to a' in G iff the value for a to a' in the causal activity matrix M exceeds some threshold τ . Note that $V \subseteq A$ since some activities in A might not be represented in G .

Definition 9 (Activity Cluster).

Let $A \subseteq \mathcal{U}_A$ be a set of activities. $\mathcal{C}(A)$ denotes the set of activity clusters over A . An activity cluster $C \in \mathcal{C}(A)$ is a subset of A , that is, $C \subseteq A$.

Definition 10 (Activity Clustering).

Let $A \subseteq \mathcal{U}_A$ be a set of activities. $\hat{\mathcal{C}}(A)$ denotes the set of activity clusterings over A . An activity clustering $\hat{C} \in \hat{\mathcal{C}}(A)$ is a set of activity clusters, that is, $\hat{C} \subseteq \mathcal{P}(A)$. A k -clustering $\hat{C} \in \hat{\mathcal{C}}(A)$ is a clustering with size k , i.e. $|\hat{C}| = k$. Let $\hat{C} \in \hat{\mathcal{C}}(A)$ be a clustering over A , the number of activities in \hat{C} is denoted by $||\hat{C}|| = \left| \bigcup_{C \in \hat{C}} C \right|$, i.e. $||\hat{C}||$ signifies the number of unique activities in \hat{C} .

3.4 Process Models and Process Discovery

Process discovery aims at discovering a model from an event log while conformance checking aims at diagnosing the differences between observed and modeled behavior (resp. the event log and the model). Literature suggests many different notations for models. We abstract from any specific model notation, but will define the set of algorithms that *discover* a model from an event log. Various discovery algorithms have been proposed in literature. These discovery algorithms are often called *mining algorithms*, or *miners* in short. For an overview of different algorithms we refer to [1].

Definition 11 (Process Model).

Let $A \subseteq \mathcal{U}_A$ be a set of activities. $\mathcal{N}(A)$ denotes the set of process models over A , irrespective of the specific notation (Petri nets, transition systems, BPMN, UML ASDs, etc.) used.

Definition 12 (Discovery Algorithm).

Let $A \subseteq \mathcal{U}_A$ be a set of activities. $\mathcal{D}(A) = \mathcal{B}(\mathcal{S}(A)) \rightarrow \mathcal{N}(A)$ denotes the set of discovery algorithms over A . A discovery algorithm $D \in \mathcal{D}(A)$ discovers a process model over A from an event log over A .

3.5 Decomposed Process Discovery

As discussed, in [5], a generic approach to decomposed process mining is proposed. In terms of decomposed process discovery, this approach can be explained as follows: Let $A \subseteq \mathcal{U}_A$ be a set of activities, and let $L \in \mathcal{B}(\mathcal{S}(A))$ be an event log over A . In order to decompose the activities in L , first a causal activity matrix $M \in \mathcal{M}(A)$ is *discovered*. Any causal activity matrix discovery algorithm $D_{CA} \in \mathcal{B}(\mathcal{S}(A)) \rightarrow \mathcal{M}(A)$ can be used. From M a causal activity graph $G \in \mathcal{G}(A)$ is *filtered* (using a specific causality threshold). By choosing the value of the causality threshold carefully, we can filter out uncommon causal relations between activities or relations of which we are unsure, for example those relations introduced by noise in the event log. Once the causal activity graph G has been constructed, an activity clustering $\hat{C} \in \hat{\mathcal{C}}(A)$ is created. Any activity clustering algorithm $AC \in \mathcal{G}(A) \rightarrow \hat{\mathcal{C}}(A)$ can be used to create the clusters. For example, the *maximal decomposition* can be used where the causal activity graph is cut across its vertices and each edge ends up in precisely one submodel. This leads to the smallest possible submodels [5]. For every cluster in the clustering, L is *filtered* to a corresponding sublog by projecting the cluster to L , i.e., for all $C \in \hat{C}$ a sublog $L|_C$ is created. A process model is *discovered* for each sublog $L|_C$. These are the submodels. Any discovery algorithm $D \in \mathcal{D}(A)$ can be used to discover the submodels. Finally, the submodels are merged into an overall model. Any merging algorithm in $\mathcal{B}(\mathcal{N}(A)) \rightarrow \mathcal{N}(A)$ can be used for this step. Currently, submodels are merged based on activity labels. Note that we have $|\hat{C}|$ clusters, sublogs and submodels, and $||\hat{C}||$ activities in the final, merged model.

4 A Better Decomposition

It is apparent that the manner in which activities are clustered has a substantial effect on required processing time, and it is possible for similarly sized clusterings (in the average cluster size) to lead to very different total processing times. As a result of the vertex-cut (*maximal*) decomposition approach [5], most activities will be in two (or more) activity clusters, leading to double (or more) work, as the clusters have a lot of overlap and causal relations between them, which might not be desirable. From the analysis results in [11] we can see that this introduces a lot of unwanted overhead, and generally reduces model quality. Also, sequences or sets of activities with high causal relations are generally easily (and thus quickly) discovered by process discovery algorithms, yet the approach will often split up these activities over different clusters. Model quality can potentially suffer from a decomposition that is too fine-grained. It might be that the sublogs

created by the approach contain too little information for the process discovery algorithm to discover a good, high quality submodel from, or that a process is split up where it shouldn't be. Merging these low-quality submodels introduces additional problems.

Hence, a *good* decomposition should (1) *maximize* the causal relations between the activities *within* each cluster in the activity clustering, (2) *minimize* the causal relations and overlap *across* the clusters and (3) have approximately *equally sized* clusters. The challenge lies in finding a good balance between these three properties.

A clustering where one cluster is a subset of another cluster is not *valid* as it would lead to double work, and would thus result in an increase in required processing time without increasing (or even decreasing) model quality. Note that this definition of a valid clustering allows for disconnected clusters, and that some activities might not be in any cluster. This is acceptable as processes might consist of disconnected parts and event logs may contain noise. However, if activities are left out some special processing might be required.

Definition 13 (Valid Clustering).

Let $A \subseteq \mathcal{U}_A$ be a set of activities. Let $\hat{C} \in \hat{\mathcal{C}}(A)$ be a clustering over A . \hat{C} is a valid clustering iff: $\hat{C} \neq \emptyset \wedge \forall_{C_1, C_2 \in \hat{C} \wedge C_1 \neq C_2} C_1 \not\subseteq C_2$. $\hat{\mathcal{C}}_V(A)$ denotes the set of valid clusterings over A .

4.1 Clustering Properties

We define decomposition quality notions in terms of clustering properties. The first clustering property we define is *cohesion*. The cohesion of an activity clustering is defined as the average cohesion of each activity cluster in that clustering. A clustering with good cohesion (cohesion ≈ 1) signifies that causal relations between activities in the same cluster are optimized, whereas bad cohesion (cohesion ≈ 0) signifies that activities with few causal relations are clustered together.

Definition 14 (Cohesion).

Let $A \subseteq \mathcal{U}_A$ be a set of activities. Let $M \in \mathcal{M}(A)$ be a causal activity matrix over A , and let $\hat{C} \in \hat{\mathcal{C}}_V(A)$ be a valid clustering over A . The cohesion of clustering \hat{C} in matrix M , denoted $Cohesion(\hat{C}, M)$ is defined as follows:

$$Cohesion(\hat{C}, M) = \frac{\sum_{C \in \hat{C}} Cohesion(C, M)}{|\hat{C}|}$$

$$Cohesion(C, M) = \frac{\sum_{c_1, c_2 \in C} \max(M(c_1, c_2), 0)}{|C \times C|}$$

The second clustering property is called *coupling*, and is also represented by a number between 0 and 1. Good coupling (coupling ≈ 1) signifies that causal relations between activities across clusters are minimized. Bad coupling (coupling ≈ 0) signifies that there are a lot of causal relations between activities in different clusters.

Definition 15 (Coupling).

Let $A \subseteq \mathcal{U}_A$ be a set of activities. Let $M \in \mathcal{M}(A)$ be a causal activity matrix over A , and let $\hat{C} \in \hat{\mathcal{C}}_V(A)$ be a valid clustering over A . The coupling of clustering \hat{C} in matrix M , denoted $\text{Coupling}(\hat{C}, M)$ is defined as follows:

$$\text{Coupling}(\hat{C}, M) = \begin{cases} 1 & \text{if } |\hat{C}| \leq 1 \\ 1 - \frac{\sum_{C_1, C_2 \in \hat{C} \wedge C_1 \neq C_2} \text{Coupling}(C_1, C_2, M)}{|\hat{C}| \cdot (|\hat{C}| - 1)} & \text{if } |\hat{C}| > 1 \end{cases}$$

$$\text{Coupling}(C_1, C_2, M) = \frac{\sum_{c_1 \in C_1, c_2 \in C_2} [\max(M(c_1, c_2), 0) + \max(M(c_2, c_1), 0)]}{2 \cdot |C_1 \times C_2|}$$

Note that the weights of the causal relations are used in the calculation of cohesion and coupling. Relations of which we are not completely sure of (or that are weak) therefore have less effect on these properties than stronger ones.

The *balance* of an activity clustering is the third property. A clustering with good balance has clusters of (about) the same size. Decomposing the activities into clusters with low balance (e.g. a k -clustering with one big cluster holding almost all of the activities and $(k - 1)$ clusters with only a few activities) will not speed up discovery or conformance checking, rendering the whole decomposition approach useless. At the same time finding a clustering with perfect balance (all clusters have the same size) will most likely split up the process / log in places that “shouldn’t be split up”, as processes generally consist out of different-sized natural parts. Balance is also represented by a number between 0 and 1, where a good balance (balance ≈ 1) signifies that all clusters are about the same size and a bad balance (balance ≈ 0) signifies that the cluster sizes differ quite a lot. This balance formula utilizes the standard deviation of the sizes of the clusters in a clustering to include the magnitude of the differences in cluster sizes. A variation of this formula using squared errors or deviations could also be used as a clustering balance measure.

Definition 16 (Balance).

Let $A \subseteq \mathcal{U}_A$ be a set of activities. Let $\hat{C} \in \hat{\mathcal{C}}_V(A)$ be a valid clustering over A . The balance of clustering \hat{C} denoted $\text{Balance}(\hat{C})$ is defined as follows:

$$\text{Balance}(\hat{C}) = 1 - \frac{2 \cdot \sigma(\hat{C})}{\|\hat{C}\|}$$

Where $\sigma(\hat{C})$ signifies the standard deviation of the sizes of the clusters in the clustering \hat{C} .

In order to assess a certain decomposition based on the clustering properties, we introduce a weighted scoring function, which grades an activity clustering with a score between 0 (bad clustering) and 1 (good clustering). A weight can be set for each clustering property, depending on their relative importance. A clustering with clustering score 1 has perfect cohesion, coupling and balance scores, on the set weighing of properties.

Definition 17 (Clustering Score).

Let $A \subseteq \mathcal{U}_A$ be a set of activities. Let $M \in \mathcal{M}(A)$ be a causal activity matrix over A , and let $\hat{C} \in \hat{\mathcal{C}}_{\mathcal{V}}(A)$ be a valid clustering over A . The clustering score (score) of clustering \hat{C} in matrix M , denoted $\text{Score}(\hat{C}, M)$ is defined as follows:

$$\begin{aligned} \text{Score}(\hat{C}, M) = & \text{Cohesion}(\hat{C}, M) \cdot \left(\frac{\text{Coh}_W}{\text{Coh}_W + \text{Cou}_W + \text{Bal}_W} \right) \\ & + \text{Coupling}(\hat{C}, M) \cdot \left(\frac{\text{Cou}_W}{\text{Coh}_W + \text{Cou}_W + \text{Bal}_W} \right) \\ & + \text{Balance}(\hat{C}) \cdot \left(\frac{\text{Bal}_W}{\text{Coh}_W + \text{Cou}_W + \text{Bal}_W} \right) \end{aligned}$$

where Coh_W , Cou_W , and Bal_W are the weights for Cohesion, Coupling, and Balance.

4.2 Recomposition of Activity Clusters

Creating a good activity clustering is essentially a graph partitioning problem. The causal activity graph needs to be partitioned in parts that have (1) good cohesion, (2) good coupling and (3) good balance. The existing *maximal* decomposition approach [5] often leads to a decomposition that is too decomposed, i.e. too fine-grained. Cohesion and balance of clusterings found by this approach are usually quite good, since all clusters consist of only a few related activities. However, coupling is inherently bad, since there's a lot of overlap in the activity clusters and there are many causal relations across clusters. This decomposition approach leads to unnecessary and unwanted overhead and potential decreased model quality. We thus want to find a possibly *non-maximal* decomposition which optimizes the three clustering properties.

Instead of applying or creating a different graph partitioning algorithm, we *recompose* the activity clusters obtained by the vertex-cut decomposition. The idea is that it is possible to create a clustering that has fewer larger clusters, requiring less processing time to discover the final model, because overhead as well as cluster overlap are reduced. Additionally, model quality is likely to increase because of the higher number of activities in the clusters and the lower coupling between clusters.

There are often many ways in which a clustering can be recomposed to the desired amount of clusters, as shown in Figure 1. We are interested in the highest quality clustering of the desired size, i.e. the clustering that has the best cohesion, coupling and balance properties. A clustering that has a high clustering score will very likely lead to such a decomposition.

In order to find a good decomposition in the form of a high-scoring clustering quickly, we propose two agglomerative hierarchical recomposition approaches, which iteratively merge clusters, reducing the size of the clustering by one each iteration.

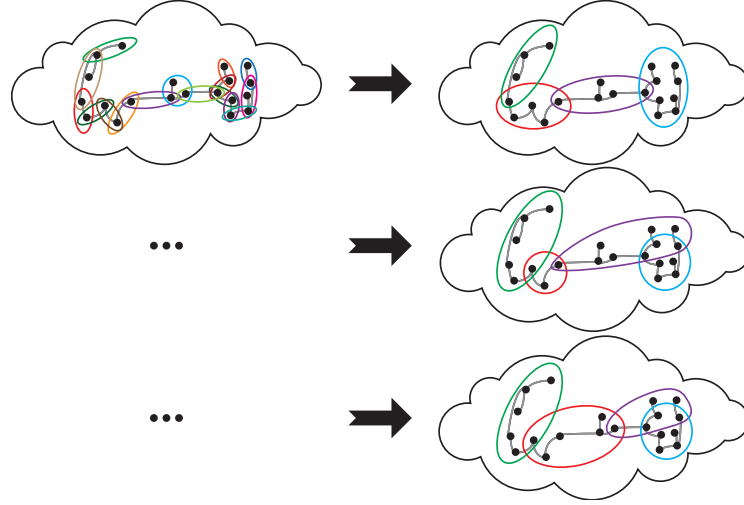


Fig. 1. 3 possible recompositions from 16 to 4 clusters. Creating a coarser clustering could potentially decrease processing time and increase model quality.

Proximity-based approach We propose an hierarchical recomposition approach based on proximity between activity clusters, where cluster coupling is used as the proximity measure. The starting point is the clustering as created by the vertex-cut approach. We repeatedly merge the clusters closest to one another (i.e. the pair of clusters with the highest coupling) until we end up with the desired amount of clusters (k). After the k -clustering is found, it is made valid by removing any clusters that are a subcluster of another cluster, if such clusters exist. It is therefore possible that the algorithm returns a clustering with size smaller than k . By merging clusters we are likely to lower the overall cohesion of the clustering. This drawback is minimized, as coupling is used as the distance measure. Coupling is also minimized. The proximity-based hierarchical recomposition approach however is less favored towards the balance property, as it is possible that -because of high coupling between clusters- two of the larger clusters are merged. In most processes however, coupling between two “original” clusters will be higher than coupling between “merged” clusters. If not, the two clusters correspond to parts of the process which are more difficult to split up (e.g. a loop, a subprocess with many interactions and/or possible paths between activities, etc.). Model quality is therefore also likely to increase by merging these clusters, as process discovery algorithms don’t have to deal with missing activities, or incorrect causal relations introduced in the corresponding sublogs. A possible downside is that as the clustering might be less balanced, processing time can be slightly higher in comparison with a perfectly-balanced decomposition.

Score-based approach We propose a second hierarchical recomposition algorithm that uses the scoring function in a look-ahead fashion. In essence, this algorithm, like the proximity-based variant, iteratively merges two clusters into one. For each combination of clusters, the score of the clustering that results from merging those clusters is calculated. The clustering with the highest score is used for the next step. The algorithm is finished when a k -clustering is reached. Like in the proximity-based approach, after the k -clustering is found, it is made valid by removing any clusters that are a subcluster of another cluster, if such clusters exist. The advantage of this approach is that specific (combinations of) clustering properties can be given priority, by setting their scoring weight(s) accordingly. For example, it is possible to distribute the activities over the clusters near perfectly, by choosing a high relative weight for balance. This would likely lead to a lower overall processing time. However, it might lead to natural parts of the process being split over multiple clusters, which could negatively affect model quality. A downside of this algorithm is that, as the algorithm only looks ahead one step, it is possible that a choice is made that ultimately leads to a lower clustering score, as that choice cannot be undone in following steps.

4.3 Implementation

All concepts and algorithms introduced in this paper are implemented in the process mining toolkit *ProM*¹, developed at the Eindhoven University of Technology. All work can be found in the *BartHompes* package². For more elaborate explanations, pseudo-code of the algorithms, and analysis results we refer to [11].

5 Use Case

The proposed recomposition techniques are tested using event logs of different sizes and properties. Results for an event log consisting of 33 unique activities, and 1000 traces are shown in this section. For this test the ILP Miner process discovery algorithm was used [17]. Discovering a model directly for this log will lead to a high quality model, but takes ~25 minutes on a modern quad-core system [11]. The vertex-cut decomposed process mining approach is able to discover a model in roughly 90 seconds, however the resulting model suffers from disconnected activities (i.e. a partitioned model). The goal is thus to find a balance between processing times and model quality.

We are interested in the clustering scores of each algorithm when recomposing the clustering created by the vertex-cut approach to a certain smaller size. Exhaustively finding the best possible clustering proved to be too time- and resource-consuming, therefore, besides the two hierarchical approaches listed here, a random recomposition approach was used which recomposes clusters randomly one million times, as to give an idea of what the best possible clustering might be. The highest found clustering score is shown on the graph. Equal

¹ See <http://www.processmining.org>

² See <https://svn.win.tue.nl/repos/prom/Packages/BartHompes/>

weights were used for the three clustering properties in order to compute the clustering scores. As can be seen in Figure 2, the vertex-cut approach creates 22 clusters. We can see that all algorithms perform very similarly in terms of clustering score. Only for very small clustering sizes the proximity-based approach performs worse than the other approaches, due to its tendency to create unbalanced clusters.

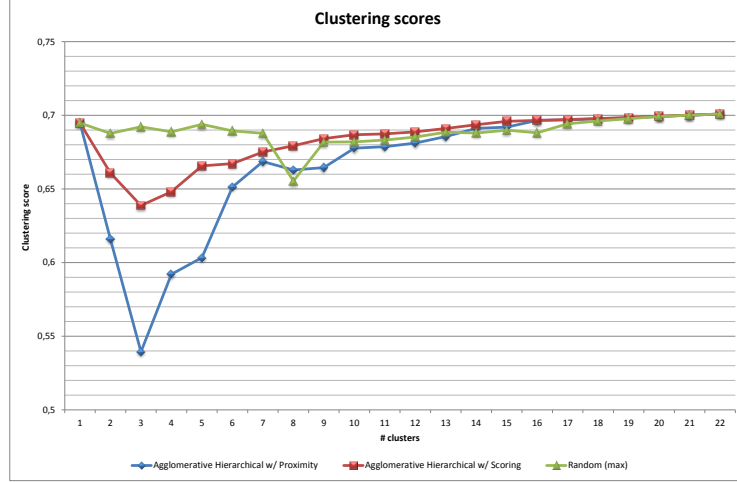


Fig. 2. Clustering score per recomposition algorithm.

Besides clustering scores, we are even more interested in how each decomposition method performs in terms of required processing time and quality of the resulting process model. In Figure 3 we can see that decomposing the event log drastically reduces processing times. For an event log this size, the decomposition steps relatively takes up negligible time (see base of bars in figure), as most time is spent discovering the submodels (light blue bars). Processing times are reduced exponentially (as expected), until a certain optimum decomposition (in terms of speed) is reached, after which overhead starts to increase time linearly again. We have included two process models (Petri Nets) discovered from the event log. Figure 4 shows the model discovered when using the vertex-cut decomposition. Figure 5 shows the model discovered when using the clustering recomposed to 11 clusters with the Proximity-based agglomerative hierarchical approach. We can see that in Figure 4, activity “10” is disconnected (marked blue). In Figure 5, this activity is connected, and a structure (loop) is discovered. We can also see that activity “12” now is connected to more activities. This shows that the vertex-cut decomposition sometimes splits up related activities, which leads to a lower quality model. By recomposing the clusters we rediscover these relations, leading to a higher quality model. Processing times for these two models are comparable, as can be seen in Figure 3.

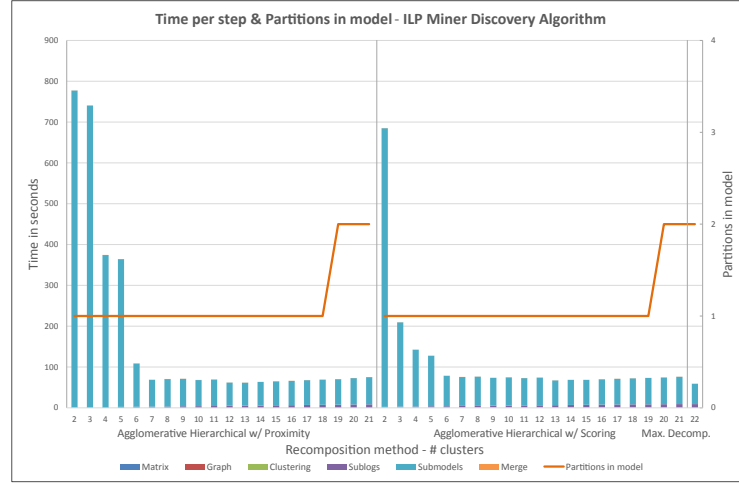


Fig. 3. Time per step & partitions in model using the Agglomerative Hierarchical recomposition approaches and the ILP Miner process discovery algorithm.

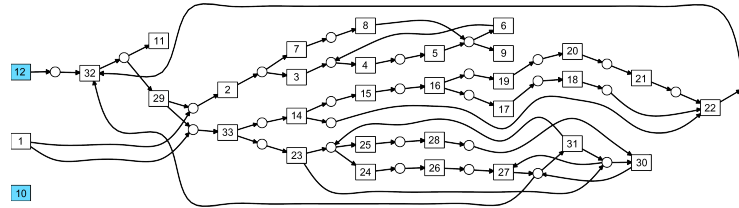


Fig. 4. Process model discovered using the vertex-cut decomposition. Some activities are disconnected in the final model.

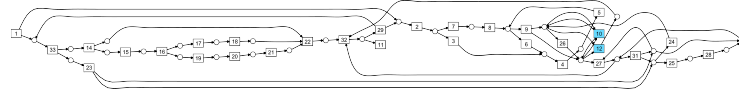


Fig. 5. Process model discovered using the vertex-cut clustering recomposed to 11 clusters. Previously disconnected activities are connected again, improving model quality.

6 Conclusions and Future Work

In decomposed process discovery, large event logs are decomposed by somehow clustering their events (activities), and there are many ways these activity clusterings can be made. Hence, good quality notions are necessary to be able to assess the quality of a decomposition before starting the time-consuming actual discovery algorithm. Being able to find a high-quality decomposition plays a key role in the success of decomposed process mining, even though the decomposition step takes relatively very little time.

By using a better decomposition, less problems arise when discovering submodels for sublogs and when merging submodels into the overall process model. We introduced three quality notions in the form of clustering properties: *cohesion*, *coupling* and *balance*. It was shown that finding a *non-maximal* decomposition can potentially lead to a decrease in required processing time while maintaining or even improving model quality, compared to the existing vertex-cut *maximal* decomposition approach. We have proposed two variants of an agglomerative hierarchical recomposition technique, which are able to create a high-quality decomposition for any given size, in very little time.

Even though the scope was limited to decomposed process discovery, the introduced quality notions and decomposition approaches can be applied to decomposed conformance checking as well. However, more work is needed to incorporate them in a conformance checking environment.

Besides finding a better decomposition, we believe improvements can be gained in finding a better, more elaborate algorithm to merge submodels into the overall process model. By simply merging submodels based on activity labels it is likely that implicit paths are introduced. Model quality in terms of fitness, simplicity, generality or precision could suffer. An additional post-processing step (potentially using causal relations) could also solve this issue.

Even though most interesting process discovery algorithms are exponential in the number of different activities, adding an infrequent or almost unrelated activity to a cluster might not increase computation time for that cluster as much as adding a frequent or highly related one. Therefore, besides weighing causal relations between activities in the causal activity matrix, activities themselves might be weighted as well. Frequency and connectedness are some of the many possible properties that can be used as weights. It might be possible that one part of a process can be discovered easily by a simple algorithm whereas another, more complex part of the process needs a more involved discovery algorithm to be modeled correctly. Further improvements in terms of processing time can be gained by somehow detecting the complexity of a single submodel in a sublog, and choosing an adequate discovery algorithm.

Finally, as discussed, the proposed recomposition algorithms expect the desired amount of clusters to be given. Even though the algorithms were shown to provide good results for any chosen number, the approach would benefit from some method that determines a fitting clustering size for a given event log. This would also mean one less potentially uncertain step for the end-user.

References

- [1] van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer, Berlin (2011) [1](#), [3](#), [4](#), [5](#)
- [2] van der Aalst, W.M.P.: Decomposing process mining problems using passages. In: Application and Theory of Petri Nets, pp. 72–91. Springer (2012) [2](#)

- [3] van der Aalst, W.M.P.: Distributed Process Discovery and Conformance Checking. In: de Lara, J., Zisman, A. (eds.) FASE. Lecture Notes in Computer Science, vol. 7212, pp. 1–25. Springer (2012) 2
- [4] van der Aalst, W.M.P.: A general divide and conquer approach for process mining. In: Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on. pp. 1–10. IEEE (2013) 3
- [5] van der Aalst, W.M.P.: Decomposing Petri nets for process mining: A generic approach. Distributed and Parallel Databases 31(4), 471–507 (2013) 2, 4, 6, 9
- [6] Bratosin, C.C., Sidorova, N., van der Aalst, W.M.P.: Distributed genetic process mining. In: Evolutionary Computation (CEC), 2010 IEEE Congress on. pp. 1–8. IEEE (2010) 2
- [7] Carmona, J.: Projection approaches to process mining using region-based techniques. Data Min. Knowl. Discov. 24(1), 218–246 (2012), <http://dblp.uni-trier.de/db/journals/datamine/datamine24.html> 2
- [8] Carmona, J., Cortadella, J., Kishinevsky, M.: A Region-Based Algorithm for Discovering Petri Nets from Event Logs. In: Business Process Management (BPM2008). pp. 358–373 (2008) 2
- [9] Carmona, J., Cortadella, J., Kishinevsky, M.: Divide-and-conquer strategies for process mining. In: Business Process Management, pp. 327–343. Springer (2009) 3
- [10] Goedertier, S., Martens, D., Vanthienen, J., Baesens, B.: Robust Process Discovery with Artificial Negative Events. Journal of Machine Learning Research 10, 1305–1340 (2009) 2
- [11] Hompes, B.F.A.: On Decomposed Process Mining: How to Solve a Jigsaw Puzzle with Friends. Master’s thesis, Eindhoven University of Technology, Eindhoven, The Netherlands (2014), <http://repository.tue.nl/776743> 6, 11
- [12] Muñoz-Gama, J., Carmona, J., van der Aalst, W.M.P.: Conformance Checking in the Large: Partitioning and Topology. In: Daniel, F., Wang, J., Weber, B. (eds.) BPM. Lecture Notes in Computer Science, vol. 8094, pp. 130–145. Springer (2013) 2
- [13] Muñoz-Gama, J., Carmona, J., van der Aalst, W.M.P.: Hierarchical Conformance Checking of Process Models Based on Event Logs. In: Colom, J.M., Desel, J. (eds.) Petri Nets. Lecture Notes in Computer Science, vol. 7927, pp. 291–310. Springer (2013) 2
- [14] Reguieg, H., Toumani, F., Motahari-Nezhad, H.R., Benatallah, B.: Using mapreduce to scale events correlation discovery for business processes mining. In: Business Process Management, pp. 279–284. Springer (2012) 2
- [15] Vanderfeesten, I.T.P.: Product-based design and support of workflow processes (2009) 3
- [16] Verbeek, H.M.W., van der Aalst, W.M.P.: Decomposed Process Mining: The ILP Case. In: BPI 2014 Workshop (2014), accepted for publication 4
- [17] van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. In: Applications and Theory of Petri Nets, pp. 368–387. Springer (2008) 11

Discovery of Frequent Episodes in Event Logs

Maikel Leemans and Wil M.P. van der Aalst

Eindhoven University of Technology, P.O. Box 513, 5600 MB, Eindhoven,
The Netherlands. m.leemans@tue.nl, w.m.p.v.d.aalst@tue.nl

Abstract. Lion’s share of process mining research focuses on the discovery of end-to-end process models describing the characteristic behavior of observed cases. The notion of a process instance (i.e., the case) plays an important role in process mining. Pattern mining techniques (such as frequent itemset mining, association rule learning, sequence mining, and traditional episode mining) do not consider process instances. An episode is a collection of partially ordered events. In this paper, we present a new technique (and corresponding implementation) that discovers frequently occurring *episodes* in event logs thereby exploiting the fact that events are associated with cases. Hence, the work can be positioned in-between process mining and pattern mining. Episode discovery has its applications in, amongst others, discovering local patterns in complex processes and conformance checking based on partial orders. We also discover episode rules to predict behavior and discover correlated behaviors in processes. We have developed a ProM plug-in that exploits efficient algorithms for the discovery of frequent episodes and episode rules. Experimental results based on real-life event logs demonstrate the feasibility and usefulness of the approach.

1 Introduction

Process mining provides a powerful way to analyze operational processes based on event data. Unlike classical purely model-based approaches (e.g., simulation and verification), process mining is driven by “raw” observed behavior instead of assumptions or aggregate data. Unlike classical data-driven approaches, process mining is truly process-oriented and relates events to high-level end-to-end process models [1].

In this paper, we use ideas *inspired by episode mining [2] and apply these to the discovery of partially ordered sets of activities in event logs*. *Event logs* serve as the starting point for process mining. An event log can be viewed as a multiset of *traces* [1]. Each trace describes the life-cycle of a particular *case* (i.e., a *process instance*) in terms of the *activities* executed. Often event logs store additional information about events, e.g., the *resource* (i.e., person or device) executing or initiating the activity, the *timestamp* of the event, or *data elements* (e.g., cost or involved products) recorded with the event.

Each trace in the event log describes the life-cycle of a case from start to completion. Hence, process discovery techniques aim to transform these event logs into *end-to-end process models*. Often the overall end-to-end process model is rather complicated because of the variability of real life processes. This results in “Spaghetti-like” diagrams. Therefore, it is interesting to also search for more local patterns in the event log – using episode discovery – while still exploiting the notion of process instances. Another useful application of episode discovery is conformance checking based on partial orders [3].

Since the seminal papers related to the Apriori algorithm [4, 5, 6], many pattern mining techniques have been proposed. These techniques do not consider the ordering of events [4] or assume an unbounded stream of events [5, 6] without considering process instances. Mannila et al. [2] proposed an extension of sequence mining [5, 6] allowing for partially ordered events. An episode is a partially ordered set of activities and it is frequent if it is “embedded” in many sliding time windows. Unlike in [2], our episode discovery technique does not use an arbitrary sliding window. Instead, we exploit the notion of process instances. Although the idea is fairly straightforward, as far as we know, this notion of frequent episodes was never applied to event logs.

Numerous applications of process mining to real-life event logs illustrate that *concurrency* is a key notion in process discovery [1, 7, 8]. One should avoid showing all observed *interleavings* in a process model. First of all, the model gets too complex (think of the classical “state-explosion problem”). Second, the resulting model will be overfitting (typically one sees only a fraction of the possible interleavings). This makes the idea of episode mining particularly attractive.

The remainder of this paper is organized as follows. Section 2 positions the work in existing literature. The novel notion of episodes and the corresponding rules are defined in Section 3. Section 4 describes the algorithms and corresponding implementation in the process mining framework *ProM*. The approach and implementation are evaluated in Section 5 using several publicly available event logs. Section 6 concludes the paper.

2 Related Work

The notion of frequent episode mining was first defined by Mannila et al. [2]. In their paper, they applied the notion of frequent episodes to (large) event sequences. The basic pruning technique employed in [2] is based on the frequency of episodes in an event sequence. Mannila et al. considered the mining of serial and parallel episodes separately, each discovered by a distinct algorithm. Laxman and Sastry improved on the episode discovery algorithm of Mannila by employing new frequency calculation and pruning techniques [9]. Experiments suggest that the improvement of Laxman and Sastry yields a 7 times speedup factor on both real and synthetic datasets.

Related to the discovery of episodes or partial orders is the discovery of end-to-end process models able to capture concurrency explicitly. The α algorithm [10] was the first process discovery algorithm adequately handling concurrency. Many other discovery techniques followed, e.g., *heuristic* mining [11] able to deal with noise and low-frequent behavior. The HeuristicsMiner is based on the notion of causal nets (C-nets). Several variants of the α algorithm have been proposed [12, 13]. Moreover, completely different approaches have been proposed, e.g., the different types of *genetic* process mining [14, 15], techniques based on *state-based regions* [16, 17], and techniques based on *language-based regions* [18, 19]. Another, more recent, approach is *inductive* process mining where the event log is split recursively [20]. The latter technique always produces a block-structured and sound process model. All the discovery techniques mentioned are able to uncover concurrency based on example behavior in the log. Additional feature comparisons are summarised in Table 1.

The episode mining technique presented in this paper is based on the discovery of frequent item sets. A well-known algorithm for mining frequent item sets and association rules is the Apriori algorithm by Agrawal and Srikant [4]. One of the

pitfalls in association rule mining is the huge number of solutions. One way of dealing with this problem is the notion of representative association rules, as described by Kryszkiewicz [21]. This notion uses user specified constraints to reduce the number of ‘similar’ results. Both sequence mining [5, 6] and episode mining [2] can be viewed as extensions of frequent item set mining.

	Exploits process instances Mines end-to-end model Soundness guaranteed Sequence Choice Concurrency Silent (τ) transitions Duplicate Activities							
Agrawal, Sequence mining [4]	-	-	n.a.	+	-	-	-	-
Manilla, Episode mining [2]	-	-	n.a.	+	-	+	-	-
Leemans M., Episode discovery	+	-	n.a.	+	-	+	-	+
Van der Aalst, α -algorithm [10]	+	+	-	+	+	+	-	-
Weijters, Heuristics mining [11]	+	+	-	+	+	+	-	-
De Medeiros, Genetic mining [14, 15]	+	+	-	+	+	+	+	+
Solé, State Regions [16, 17]	+	+	-	+	+	+	-	-
Bergenthum, Language Regions [18, 19]	+	+	-	+	+	+	-	-
Leemans S.J.J., Inductive [20]	+	+	+	+	+	+	+	-

Table 1. Feature comparison of discussed discovery algorithms

3 Event Logs, Episodes, and Episode Rules

This section defines basic notions such as event logs, episodes and rules. Note that our notion of episodes is different from the notion in [2] which does not consider process instances.

3.1 Event Logs

Activities and Traces Let \mathcal{A} be the alphabet of activities. A trace is a list (sequence) $T = \langle A_1, \dots, A_n \rangle$ of activities $A_i \in \mathcal{A}$ occurring at time index i relative to the other activities in T .

Event log An event log $L = [T_1, \dots, T_m]$ is a multiset of traces T_i . Note that the same trace may appear multiple times in an event log. Each trace corresponds to an execution of a process, i.e., a *case* or *process instance*. In this simple definition of an event log, an event refers to just an *activity*. Often event logs store additional information about events, such as *timestamps*.

3.2 Episodes

Episode An episode is a partial ordered collection of events. Episodes are depicted using the transitive reduction of directed acyclic graphs, where the nodes represent events, and the edges imply the partial order on events. Note that the presence of an edge implies serial behavior. Figure 1 shows the transitive reduction of an example episode.

Formally, an episode $\alpha = (V, \leq, g)$ is a triple, where V is a set of events (nodes), \leq is a partial order on V , and $g : V \mapsto \mathcal{A}$ is a left-total function from events to activities, thereby labelling the nodes/events [2]. For two vertices $u, v \in V$ we have $u < v$ iff $u \leq v$ and $u \neq v$. In addition, we define G to be the multiset of activities/labels used: $G = [g(v) \mid v \in V]$. Note that if $|V| \leq 1$, then we got an singleton or empty episode. For the rest of this paper, we ignore empty episodes. We call an episode *parallel* when $\leq = \emptyset$.

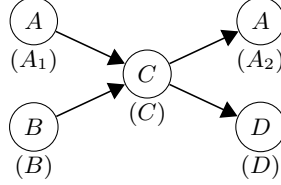


Fig. 1. Shown is the transitive reduction of the partial order for an example episode. The circles represent nodes (events), with the activity labelling imposed by g inside the circles, and an event ID beneath the nodes in parenthesis. In this example, events A_1 and B can happen in parallel (as can A_2 and D), but event C can only happen after both A_1 and B have occurred.

Subepisode and Equality An episode $\beta = (V', \leq', g')$ is a subepisode of $\alpha = (V, \leq, g)$, denoted $\beta \preceq \alpha$, iff there is an injective mapping $f : V' \mapsto V$ such that:

$$(\forall v \in V' : g'(v) = g(f(v))) \wedge (\forall v, w \in V' \wedge v \leq' w : f(v) \leq f(w))$$

An episode β equals episode α , denoted $\beta = \alpha$ iff $\beta \preceq \alpha \wedge \alpha \preceq \beta$. An episode β is a strict subepisode of α , denoted $\beta \prec \alpha$, iff $\beta \preceq \alpha \wedge \beta \neq \alpha$.

Episode construction Two episodes $\alpha = (V, \leq, g)$ and $\beta = (V', \leq', g')$ can be ‘merged’ to construct a new episode $\gamma = (V^*, \leq^*, g^*)$. $\alpha \oplus \beta$ is the smallest γ (i.e., smallest sets V^* and \leq^*) such that $\alpha \preceq \gamma$ and $\beta \preceq \gamma$. As shown below, such an episode γ always exists.

The smallest sets criteria implies that every event $v \in V^*$ and ordered pair $v, w \in V^* \wedge v \leq^* w$ must have a witness in α and/or β . Formally, $\gamma = \alpha \oplus \beta$ iff there exists injective mappings $f : V \mapsto V^*$ and $f' : V' \mapsto V^*$ such that:

$$\begin{aligned} G^* &= G \cup G' && \text{activity witness} \\ \leq^* &= \{ (f(v), f(w)) \mid (v, w) \in \leq \} \cup \{ (f'(v), f'(w)) \mid (v, w) \in \leq' \} && \text{order witness} \end{aligned}$$

Occurrence An episode $\alpha = (V, \leq, g)$ occurs in an event trace $T = \langle A_1, \dots, A_n \rangle$, denoted $\alpha \sqsubseteq T$, iff there exists an injective mapping $h : V \mapsto \{1, \dots, n\}$ such that:

$$(\forall v \in V : g(v) = A_{h(v)}) \wedge (\forall v, w \in V \wedge v \leq w : h(v) \leq h(w))$$

In Figure 2 an example of an “event to trace map” h for occurrence checking is given.

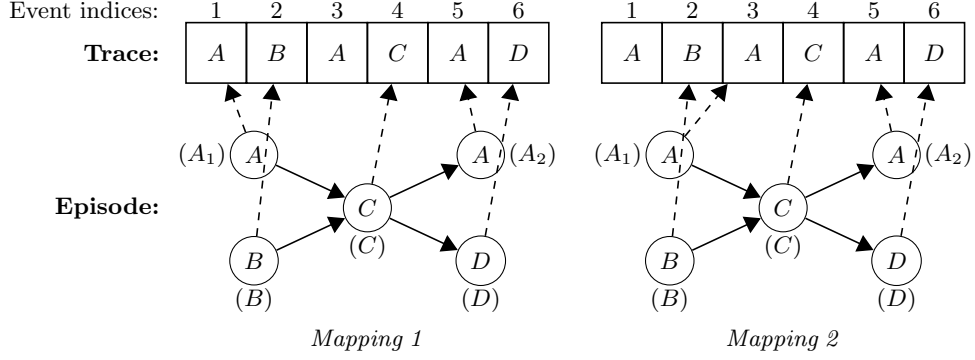


Fig. 2. Shown are two possible mappings h (the dotted arrows) for checking occurrence of the example episode in a trace. The shown graphs are the transitive reduction of the partial order of the example episode. Note that with the left mapping (*Mapping 1*) also an episode with the partial order $A_1 < B$ occurs in the given trace, in the right mapping (*Mapping 2*) the same holds for an episode with the partial order $B < A_1$.

Frequency The frequency $\text{freq}(\alpha)$ of an episode α in an event log $L = [T_1, \dots, T_m]$ is defined as:

$$\text{freq}(\alpha) = \frac{|\{T_i \mid T_i \in L \wedge \alpha \sqsubseteq T_i\}|}{|L|}$$

Given a frequency threshold minFreq , an episode α is frequent iff $\text{freq}(\alpha) \geq \text{minFreq}$. During the actual episode discovery, we use the fact given in Lemma 1.

Lemma 1 (Frequency and subepisodes). *If an episode α is frequent in an event log L , then all subepisodes β with $\beta \preceq \alpha$ are also frequent in L . Formally, we have for a given α :*

$$(\forall \beta \preceq \alpha : \text{freq}(\beta) \geq \text{freq}(\alpha))$$

Activity Frequency The activity frequency $\text{ActFreq}(A)$ of an activity $A \in \mathcal{A}$ in an event log $L = [T_1, \dots, T_m]$ is defined as:

$$\text{ActFreq}(A) = \frac{|\{T_i \mid T_i \in L \wedge A \in T_i\}|}{|L|}$$

Given a frequency threshold minActFreq , an activity A is frequent iff $\text{ActFreq}(A) \geq \text{minActFreq}$.

Trace Distance Given episode $\alpha = (V, \leq, g)$ occurring in an event trace $T = \langle A_1, \dots, A_n \rangle$, as indicated by the event to trace map $h : V \mapsto \{1, \dots, n\}$. Then the trace distance $\text{traceDist}(\alpha, T)$ is defined as:

$$\text{traceDist}(\alpha, T) = \max \{ h(v) \mid v \in V \} - \min \{ h(v) \mid v \in V \}$$

In Figure 2, the left mapping yields $\text{traceDist}(\alpha, T) = 6 - 1 = 5$, and the right mapping yields $\text{traceDist}(\alpha, T) = 6 - 2 = 4$.

Given a trace distance interval $[minTraceDist, maxTraceDist]$, an episode α is accepted in trace T with respect to the trace distance interval iff $minTraceDist \leq traceDist(\alpha, T) \leq maxTraceDist$.

Informally, the conceptual idea behind a trace distance interval is that we are interested in a partial order on events occurring relatively close in time.

3.3 Episode Rules

Episode rule An episode rule is an association rule $\beta \Rightarrow \alpha$ with $\beta \prec \alpha$ stating that after seeing β , then likely the larger episode α will occur as well.

The confidence of the episode rule $\beta \Rightarrow \alpha$ is given by:

$$conf(\beta \Rightarrow \alpha) = \frac{freq(\alpha)}{freq(\beta)}$$

Given a confidence threshold $minConf$, an episode rule $\beta \Rightarrow \alpha$ is valid iff $conf(\beta \Rightarrow \alpha) \geq minConf$. During the actual episode rule discovery, we use Lemma 2.

Lemma 2 (Confidence and subepisodes). *If an episode rule $\beta \Rightarrow \alpha$ is valid in an event log L , then for all episodes β' with $\beta \prec \beta' \prec \alpha$ the event rule $\beta' \Rightarrow \alpha$ is also valid in L . Formally:*

$$(\forall \beta \prec \beta' \prec \alpha : conf(\beta \Rightarrow \alpha) \leq conf(\beta' \Rightarrow \alpha))$$

Episode rule magnitude Let the graph size $size(\alpha)$ of an episode α be denoted as the sum of the nodes and edges in the transitive reduction of the episode. The magnitude of an episode rule is defined as:

$$mag(\beta \Rightarrow \alpha) = \frac{size(\beta)}{size(\alpha)}$$

Intuitively, the magnitude of an episode rule $\beta \Rightarrow \alpha$ represents how much episode α ‘adds to’ or ‘magnifies’ episode β . The magnitude of an Episode rule allows smart filtering on generated rules. Typically, an extremely low (approaching zero) or high (approaching one) magnitude indicates a trivial episode rule.

4 Realization

The definitions and insights provided in the previous section have been used to implement a episode (rule) discovery plug-in in *ProM*. To be able to analyze real-life event logs, we need efficient algorithms. These are described next.

Notation: in the listed algorithms, we will reference to the elements of an episode $\alpha = (V, \leq, g)$ as $\alpha.V$, $\alpha.\leq$ and $\alpha.g$.

4.1 Frequent Episode Discovery

Discovering frequent episodes is done in two phases. The first phase discovers parallel episodes (i.e., nodes only), the second phase discovers partial orders (i.e., adding the edges). The main routine for discovering frequent episodes is given in Algorithm 1.

Algorithm 1: Episodes discovery

Input: An event log L , an activity alphabet \mathcal{A} , a frequency threshold $minFreq$.

Output: A set of frequent episodes Γ

Description: Two-phase episode discovery. Each phase alternates by generating new candidate episodes (C_l), and recognizing frequent candidates in the event log (F_l).

Proof of termination: Note that candidate episode generation with $F_l = \emptyset$ will yield $C_l = \emptyset$. Since each iteration the generated episodes become strictly larger (in terms of V and \leq), eventually the generated episodes cannot occur in any trace. Therefore, always eventually $F_l = \emptyset$, and thus we will always terminate.

EPISODEDISCOVERY($L, \mathcal{A}, minFreq$)

```

(1)    $\Gamma = \emptyset$ 
(2)   // Phase 1: discover parallel episodes
(3)    $l = 1$  // Tracks the number of nodes
(4)    $C_l = \{ (V, \leq = \emptyset, g = \{v \mapsto a\}) \mid |V| = 1 \wedge v \in V \wedge a \in \mathcal{A} \}$ 
(5)   while  $C_l \neq \emptyset$ 
(6)      $F_l = \text{RECOGNIZEFREQUENTEPISODES}(L, C_l, minFreq)$ 
(7)      $\Gamma = \Gamma \cup F_l$ 
(8)      $C_l = \text{GENERATECANDIDATEPARALLEL}(l, F_l)$ 
(9)      $l = l + 1$ 
(10)  // Phase 2: discover partial orders
(11)   $l = 1$  // Tracks the number of edges
(12)   $C_l = \{ (V = \gamma.V, \leq = \{(v, w)\}, g = \gamma.g) \mid \gamma \in \Gamma \wedge v, w \in \gamma.V \wedge v \neq w \}$ 
(13)  while  $C_l \neq \emptyset$ 
(14)     $F_l = \text{RECOGNIZEFREQUENTEPISODES}(L, C_l, minFreq)$ 
(15)     $\Gamma = \Gamma \cup F_l$ 
(16)     $C_l = \text{GENERATECANDIDATEORDER}(l, F_l)$ 
(17)     $l = l + 1$ 
(18)  return  $\Gamma$ 
```

4.2 Episode Candidate Generation

The generation of candidate episodes for each phase is an adaptation of the well-known Apriori algorithm over an event log. Given a set of frequent episodes F_l , we can construct a candidate episode γ by combining two partially overlapping episodes α and β from F_l . Note that this implements the episode construction operation $\gamma = \alpha \oplus \beta$.

For phase 1, we have F_l contains frequent episodes with l nodes and no edges. A candidate episode γ will have $l + 1$ nodes, resulting from episodes α and β that overlap on the first $l - 1$ nodes. This generation is implemented by Algorithm 2.

For phase 2, we have F_l contains frequent episodes with l edges. A candidate episode γ will have $l + 1$ edges, resulting from episodes α and β that overlap on the first $l - 1$ edges and have the same set of nodes. This generation is implemented by Algorithm 3. Note that, formally, the partial order \leq is the transitive closure of the set of edges being constructed, and that the edges are really only the transitive reduction of this partial order.

Algorithm 2: Candidate episode generation – Parallel

Input: A set of frequent episodes F_l with l nodes.
Output: A set of candidate episodes C_{l+1} with $l + 1$ nodes.
Description: Generates candidate episodes γ by merging overlapping episodes α and β (i.e., $\gamma = \alpha \oplus \beta$). For parallel episodes, overlapping means: sharing $l - 1$ nodes.
 GENERATECANDIDATEPARALLEL(l, F_l)

```

(1)    $C_{l+1} = \emptyset$ 
(2)   for  $i = 0$  to  $|F_l| - 1$ 
(3)     for  $j = i + 1$  to  $|F_l| - 1$ 
(4)        $\alpha = F_l[i]$ 
(5)        $\beta = F_l[j]$ 
(6)       if  $\forall 0 \leq i \leq l - 2 : \alpha.g(\alpha.V[i]) = \beta.g(\beta.V[i])$ 
(7)          $\gamma = (V = (\alpha.V[0..l-1] \cup \beta.V[l-1]), \leq = \emptyset, g = \alpha.g \cup \beta.g)$ 
(8)          $C_{l+1} = C_{l+1} \cup \{\gamma\}$ 
(9)       else
(10)        break
(11)  return  $C_{l+1}$ 

```

Algorithm 3: Candidate episode generation – Partial order

Input: A set of frequent episodes F_l with l edges.
Output: A set of candidate episodes C_{l+1} with $l + 1$ edges.
Description: Generates candidate episodes γ by merging overlapping episodes α and β (i.e., $\gamma = \alpha \oplus \beta$). For partial order episodes, overlapping means: sharing all nodes and $l - 1$ edges.
 GENERATECANDIDATEORDER(l, F_l)

```

(1)    $C_{l+1} = \emptyset$ 
(2)   for  $i = 0$  to  $|F_l| - 1$ 
(3)     for  $j = i + 1$  to  $|F_l| - 1$ 
(4)        $\alpha = F_l[i]$ 
(5)        $\beta = F_l[j]$ 
(6)       if  $\alpha.V = \beta.V \wedge \alpha.g = \beta.g \wedge \alpha.\leq[0..l-2] = \beta.\leq[0..l-2]$ 
(7)          $\gamma = (V = \alpha.V, \leq = (\alpha.E[0..l-1] \cup \beta.E[l-1]), g = \alpha.g)$ 
(8)          $C_{l+1} = C_{l+1} \cup \{\gamma\}$ 
(9)       else
(10)        break
(11)  return  $C_{l+1}$ 

```

4.3 Frequent Episode Recognition

In order to check if a candidate episode α is frequent, we check if $\text{freq}(\alpha) \geq \text{minFreq}$. The computation of $\text{freq}(\alpha)$ boils down to counting the number of traces T with $\alpha \sqsubseteq T$. Algorithm 4 recognizes all frequent episodes from a set of candidate episodes using the above described approach. Note that for both parallel and partial order episodes we can use the same recognition algorithm.

Algorithm 4: Recognize frequent episodes

Input: An event log L , a set of candidate episodes C_l , a frequency threshold minFreq .
Output: A set of frequent episodes F_l .
Description: Recognizes frequent episodes, by filtering out candidate episodes that do not occur frequently in the log. **Note:** If $F_l = \emptyset$, then $C_l = \emptyset$.
 RECOGNIZEFREQUENTEPISODES($L, C_l, \text{minFreq}$)

```

(1)    $\text{support} = [0, \dots, 0]$  with  $|\text{support}| = |C_l|$ 
(2)   foreach  $T \in L$ 
(3)     for  $i = 0$  to  $|C_l| - 1$ 
(4)       if OCCURS( $C_l[i], T$ ) then  $\text{support}[i] = \text{support}[i] + 1$ 
(5)    $F_l = \emptyset$ 
(6)   for  $i = 0$  to  $|C_l| - 1$ 
(7)     if  $\frac{\text{support}[i]}{|L|} \geq \text{minFreq}$  then  $F_l = F_l \cup \{C_l[i]\}$ 
(8)   return  $F_l$ 

```

Checking whether an episode α occurs in a trace $T = \langle A_1, \dots, A_n \rangle$ is done via checking the existence of the mapping $h : \alpha.V \mapsto \{1, \dots, n\}$. This results in checking the two propositions shown below. Algorithm 5 implements these checks.

- Checking whether each node $v \in \alpha.V$ has a unique witness in trace T .
- Checking whether the (injective) mapping h respects the partial order indicated by $\alpha.\leq$.

For the discovery of an injective mapping h for a specific episode α and trace T we use the following recipe. First, we declare the class of models $H : \mathcal{A} \mapsto \mathcal{P}(\mathbb{N})$ such that for each activity $a \in \mathcal{A}$ we get the set of indices i at which $a = A_i \in T$. Next, we try all possible models derivable from H . A model $h : \alpha.V \mapsto \{1, \dots, n\}$ is derived from H by choosing an index $i \in H(f(v))$ for each node $v \in \alpha.V$. With such a model h , we can perform the actual partial order check against $\alpha.\leq$.

Algorithm 5: This algorithm implements occurrence checking via recursive discovery of the injective mapping h as per the occurrence definition.

Input: An episode α , a trace T .

Output: True iff $\alpha \sqsubseteq T$

Description: Implements occurrence checking based on finding an occurrence proof in the form of a mapping $h : \alpha.V \mapsto \{1, \dots, n\}$.

$\text{OCCURS}(\alpha = (V, \leq, g), T)$

(1) **return** $\text{CHECKMODEL}(\alpha, \{a \mapsto \{i \mid a = A_i \in T\} \mid a \in \mathcal{A}\}, \emptyset)$

Input: An episode α , a class of mappings $H : \mathcal{A} \mapsto \mathcal{P}(\mathbb{N})$, and an intermediate mapping $h : \alpha.V \mapsto \{1, \dots, n\}$.

Output: True iff there is a mapping h , as per the occurrence definition, derivable from H

Description: Recursive implementation for finding h based on the following induction principle: Base case (*if*-part): Every $v \in V$ is mapped ($v \in \text{dom } h$). Step case (*else*-part): (IH) n vertices are mapped, step by adding a mapping for a vertex $v \notin \text{dom } h$. (I.e., induction to the number of mapped vertices.)

$\text{CHECKMODEL}(\alpha = (V, \leq, g), H, h)$

(1) **if** $\forall v \in V : v \in \text{dom } h$

(2) **return** $(\forall (v, w) \in \leq : h(v) \leq h(w))$

(3) **else**

(4) **pick** $v \in V$ with $v \notin \text{dom } h$

(5) **return** $(\exists i \in H(g(v)) :$

$\text{CHECKMODEL}(\alpha, H[g(v) \mapsto H(g(v)) \setminus \{i\}], h[v \mapsto i])$)

4.4 Pruning

Using the pruning techniques described below, we reduce the number of generated episodes (and thereby computation time and memory requirements) and filter out uninteresting results. These techniques eliminate less interesting episodes by ignoring infrequent activities and skipping partial orders on activities with low temporal locality.

Activity Pruning Based on the frequency of an activity, uninteresting episodes can be pruned in an early stage. This is achieved by replacing the activity alphabet \mathcal{A} by $\mathcal{A}^* \subseteq \mathcal{A}$, with

$(\forall A \in \mathcal{A}^* : \text{ActFreq}(A) \geq \text{minActFreq})$, on line 4 in Algorithm 1. This pruning technique allows the episode discovery algorithm to be more resistant to logs with many infrequent activities, which are indicative of exceptions or noise.

Trace Distance Pruning The pruning of episodes based on a trace distance interval can be achieved by adding the trace distance interval check to line 2 of Algorithm 5. Note that if there are two or more interpretations for h , with one passing and one rejected by the interval check, then we will find the correct interpretation thanks to the \exists on line 5.

4.5 Episode Rule Discovery

The discovery of episode rules is done after discovering all the frequent episodes. For all frequent episodes α , we consider all frequent subepisodes β with $\beta \prec \alpha$ for the episode rule $\beta \Rightarrow \alpha$.

For efficiently finding potential frequent subepisodes β , we use the notion of “discovery tree”, based on episode construction. Each time we recognize a frequent episode β created from combining frequent episodes γ and ε , we recognize β as a child of γ and ε . Similarly, γ and ε are the parents of β . See Figure 3 for an example of a discovery tree.

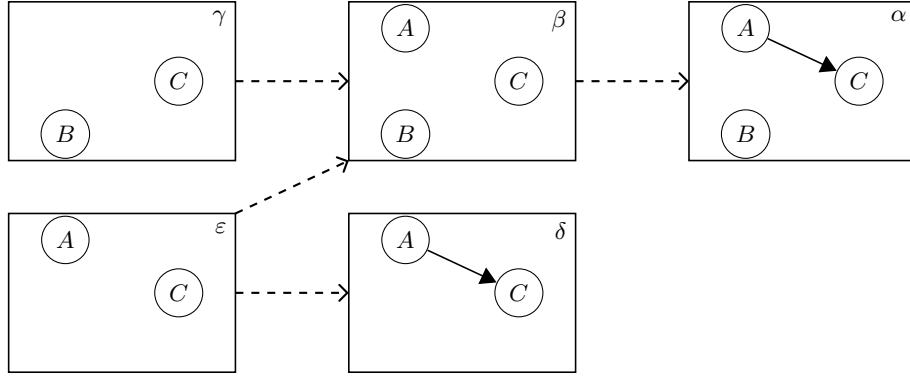


Fig. 3. Part of an example discovery tree. Each block denotes an episode. The dashed arrows between blocks denote a parent-child relationship. In this example we have, amongst others: $\beta \prec \alpha$, $\varepsilon \prec \beta$, $\varepsilon \prec \delta$ and $\delta \prec \alpha$ (not shown as a parent-child relation).

Using the discovery tree we can walk from an episode α along the discovery parents of α . Each time we find a parent β with $\beta \prec \alpha$, we can consider the parents and children of β . As result of Lemma 2, we cannot apply pruning in either direction of the parent-child relation based on the confidence $\text{conf}(\beta \Rightarrow \alpha)$. This is easy to see for the child direction. For the parent direction, observe the discovery tree in Figure 3 and $\delta \prec \alpha$. If for episode α we would stop before visiting the parents of β , we would never consider δ (which has $\delta \prec \alpha$).

4.6 Implementation Considerations

We implemented the episode discovery algorithm as a ProM 6 plug-in (see also Figure 4), written in Java. Since the OCCURS() algorithm (5) is the biggest bottleneck, this part of the implementation was considerably optimized.

5 Evaluation

This section reviews the feasibility of the approach using both synthetic and real-life event data.

5.1 Methodology

We ran a series of experiments on two type of event logs. The first event log, *bigger-example.xes*, is an artificial event log from the Chapter 5 of [1] and available via http://www.processmining.org/event_logs_and_models_used_in_book. The second event log, *BPI_Challenge_2012.xes*, is a real life event log available via doi:10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f. For these experiments we used a laptop with a Core i5-3570K CPU, 8 GB RAM and Java SE Runtime Environment 1.7.0.07-b11 (32 bit).

5.2 Performance Results

Table 2 some key characteristics for both event logs. We examined the effects of the parameters *minFreq*, *minActFreq* and *maxTraceDist* on the running time and the discovered number of episodes. In Figure 4 an indication (screenshots) of the ProM plugin output is given.

	# traces	Avg. events/trace	Min. events/trace	Max. events/trace
bigger-example.xes	1391	5	5	17
BPI_Challenge_2012.xes	13087	20	3	175

Table 2. Metadata for the used event logs

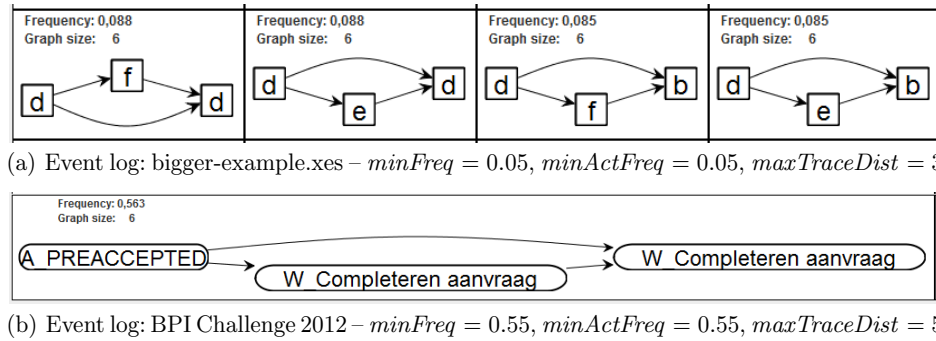


Fig. 4. Screenshots of the results in the ProM plugin. Shown are the transitive reductions of the discovered episodes. Note that in the episodes in Figure 4(a), multiple nodes are allowed to have the same label.

As can be seen in all the experiments in Figure 5, we see that the running time is strongly related to the discovered number of episodes. Note that if some parameters

are poorly chosen, like high *maxTraceDist* in Figure 5(f), then a relatively large class of episodes seems to become frequent, thus increasing the running time drastically.

For a reasonably low number of frequent episodes (< 500 , more will a human not inspect), the algorithm turns out to be quite fast (at most a few seconds for the Challenge log). We noted a virtual nonexistent contribution of the parallel episode mining phase to the total running time. This can be explained by a simple combinatorial argument: there are far more partial orders to be considered than there are parallel episodes.

An analysis of the effects of changing the *minFreq* parameter (Figure 5(a), 5(b)) shows that a poorly chosen value results in many episodes. In addition, the *minFreq* parameter gives us fine-grained control of the number of results. It gradually increases the total number of episodes for lower values. Note that, especially for the Challenge event log, low values for *minFreq* can dramatically increase the running time. This is due to the large number of candidate episodes being generated.

Secondly, note that for the *minActFreq* parameter (Figure 5(c), 5(d)), there seems to be a cutoff point that separates frequent from infrequent activities. Small changes around this cutoff point may have a dramatic effect on the number of episodes discovered.

Finally, for the *maxTraceDist* parameter (Figure 5(e), 5(f)), we see that this parameter seems to have a sweet-spot where a low – but not too low – number of episodes are discovered. Chosen a value for *maxTraceDist* just after this sweet-spot yields a huge number of episodes.

When comparing the artificial and real life event logs, we see a remarkable pattern. The artificial event log (*bigger-example.xes*), shown in Figure 5(a) appears to be far more fine-grained than the real life event log (*BPI_Challenge_2012.xes*) shown in Figure 5(b). In the real life event log there appears to be a clear distinction between frequent and infrequent episodes. In the artificial event log a more exponential pattern occurs. Most of the increase in frequent episodes, for decreasing *minFreq*, is again in the partial order discovery phase.

5.3 Comparison to existing discovery algorithms

As noted in the introduction, often the overall end-to-end process models are rather complicated. Therefore, the search for local patterns (i.e., episodes) is interesting. A good example of a complicated process is the *BPI Challenge 2012* log. In Figure 6 part of the “spaghetti-like” process models are shown, as an indication of the complexity. The episodes discovered over same log, depicted in Figure 4(b) gives us a simple and clear insight into important local patterns in the *BPI Challenge 2012* log. Hence, in these “spaghetti-like” process models, the episode discovery technique allows us to quickly understand the main patterns.

6 Conclusion and Future work

In this paper, we considered the problem of discovering frequently occurring episodes in an event log. An episode is a collection of events that occur in a given partial order. We presented efficient algorithms for the discovery of frequent episodes and episode rules occurring in an event log, and presented experimental results.

Our experimental evaluation shows that the running time is strongly related to the discovered number of episodes. For a reasonably low number of frequent episodes

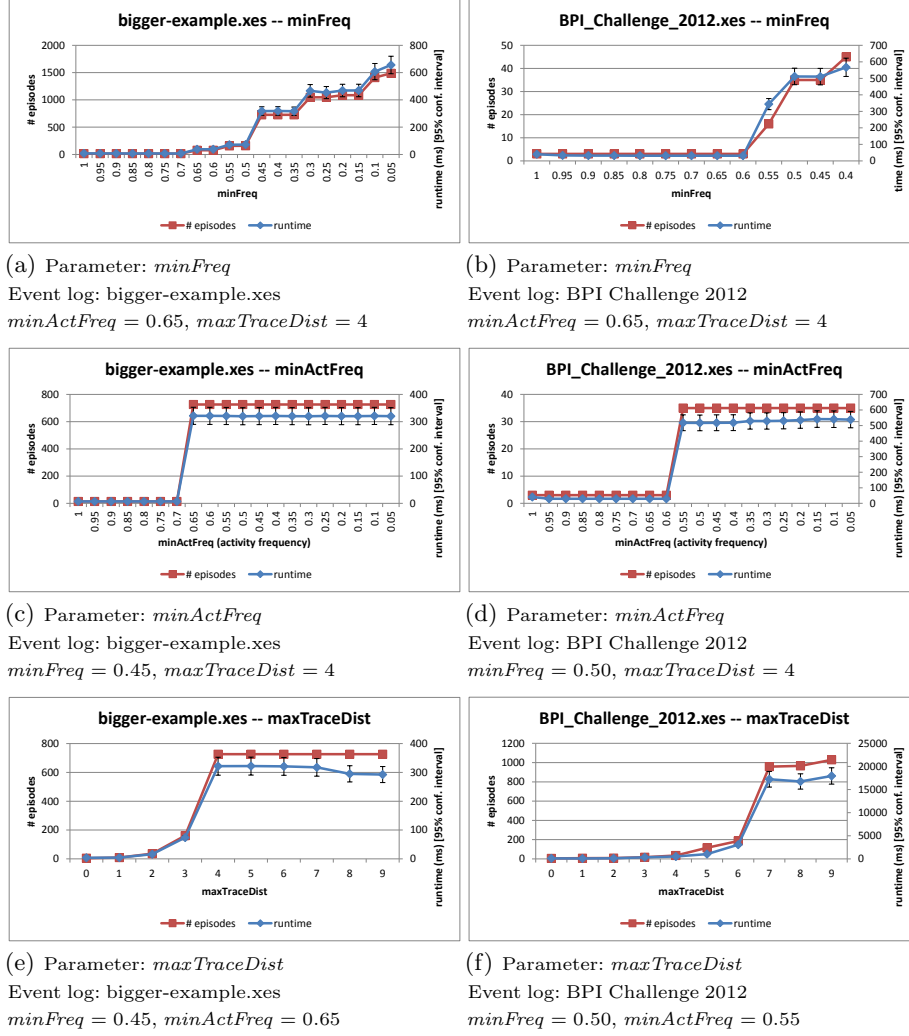


Fig. 5. Effects of the parameter on the performance and number of discovered episodes.

(< 500, more will a human not inspect), the algorithm turns out to be quite fast (at most a few seconds). The main problem is the correct setting of the episode pruning parameters *minFreq*, *minActFreq*, and *maxTraceDist*.

During the development of the algorithm for ProM 6, special attention was paid to optimizing the OCCURS() algorithm (Algorithm 5) implementation, which proved to be the main bottleneck. Future work could be to prune occurrence checking based on the parents of an episode, leveraging the fact that an episode cannot occur in a trace if a parent also did occur in that trace.

Another approach to improve the algorithm is to apply the *generic divide and conquer approach for process mining*, as defined in [22]. This approach splits the set of activities into a collection of partly overlapping activity sets. For each activity set, the log is projected onto the relevant events, and the regular episode discovery algorithm is applied. In essence, the same trick is applied as used by the *minActFreq*

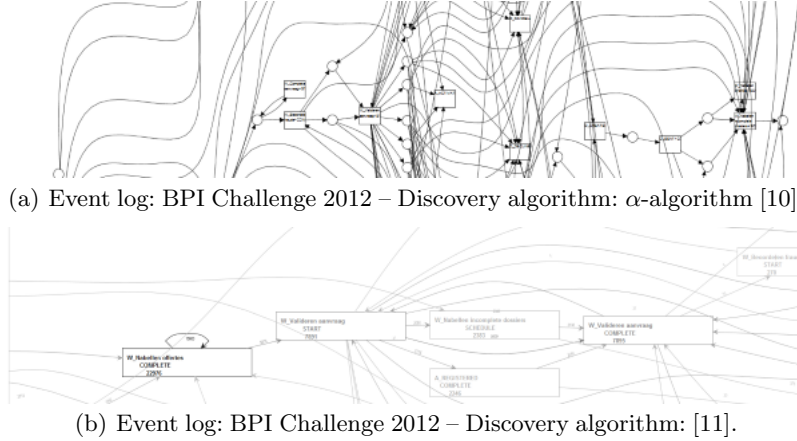


Fig. 6. Screenshots of results in other ProM plugin. Shown are parts of the Petri-nets mined with the α -algorithm and the heuristics miner.

parameter (using an alphabet subset), which is to create a different set of initial 1-node parallel episodes to start discovering with.

The main bottleneck is the frequency computation by checking the occurrence of each episode in each trace. Typically, we have a small amount of episodes to check, but many traces to check against. Using the MapReduce programming model developed by Dean and Ghemawat, we can easily parallelize the episode discovery algorithm and execute it on a large cluster of commodity machines [23]. The MapReduce programming model requires us to define *map* and *reduce* functions. The *map* function, in our case, accepts a trace and produces [episode, trace] pairs for each episode occurring in the given trace. The *reduce* function accepts an episode plus a list of traces in which that episode occurs, and outputs a singleton list if the episode is frequent, and an empty list otherwise. This way, the main bottleneck of the algorithm is effectively parallelized.

References

- [1] van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer-Verlag, Berlin (2011)
- [2] Mannila, H., Toivonen, H., Verkamo, A.I.: Discovery of Frequent Episodes in Event Sequences. *Data Mining and Knowledge Discovery* **1**(3) (1997) 259–289
- [3] Lu, X., Fahland, D., van der Aalst, W.M.P.: Conformance checking based on partially ordered event data. To appear in *Business Process Intelligence 2014, workshop SBS* (2014)
- [4] Agrawal, R., Srikant, R.: Fast Algorithms for Mining Association Rules in Large Databases. In: *Proceedings of the 20th International Conference on Very Large Data Bases. VLDB '94*, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (1994) 487–499
- [5] Agrawal, R., Srikant, R.: Mining Sequential Patterns. In: *Proceedings of the Eleventh International Conference on Data Engineering. ICDE '95*, Washington, DC, USA, IEEE Computer Society (1995) 3–14
- [6] Srikant, R., Agrawal, R.: Mining Sequential Patterns: Generalization and Performance Improvements. In: *Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology. EDBT '96*, London, UK, UK, Springer-Verlag (1996) 3–17

- [7] Lu, X., Mans, R.S., Fahland, D., van der Aalst, W.M.P.: Conformance checking in healthcare based on partially ordered event data. To appear in *Emerging Technologies and Factory Automation 2014, workshop M2H* (2014)
- [8] Fahland, D., van der Aalst, W.M.P.: Repairing process models to reflect reality. In: *Proceedings of the 10th International Conference on Business Process Management. BPM'12, Berlin, Heidelberg, Springer-Verlag* (2012) 229–245
- [9] Laxman, S., Sastry, P.S., Unnikrishnan, K.P.: Fast Algorithms for Frequent Episode Discovery in Event Sequences. In: *Proc. 3rd Workshop on Mining Temporal and Sequential Data*. (2004)
- [10] van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering* **16**(9) (2004) 1128–1142
- [11] Weijters, A.J.M.M., van der Aalst, W.M.P., de Medeiros, A.K.A.: Process Mining with the Heuristics Miner-algorithm. *BETA Working Paper Series, WP 166*, Eindhoven University of Technology, Eindhoven (2006)
- [12] de Medeiros, A.K.A., van der Aalst, W.M.P., Weijters, A.J.M.M.: Workflow mining: Current status and future directions. In Meersman, R., Tari, Z., Schmidt, C.D., eds.: *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE. Volume 2888 of Lecture Notes in Computer Science*. Springer Berlin Heidelberg (2003) 389–406
- [13] Wen, L., van der Aalst, W.M.P., Wang, J., Sun, J.: Mining process models with non-free-choice constructs. *Data Mining and Knowledge Discovery* **15**(2) (2007) 145–180
- [14] de Medeiros, A.K.A., Weijters, A.J.M.M., van der Aalst, W.M.P.: Genetic Process Mining: An Experimental Evaluation. *Data Mining and Knowledge Discovery* **14**(2) (2007) 245–304
- [15] Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery. In Meersman, R., Rinderle, S., Dadam, P., Zhou, X., eds.: *OTM Federated Conferences, 20th International Conference on Cooperative Information Systems (CoopIS 2012)*. Volume 7565 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin (2012) 305–322
- [16] Solé, M., Carmona, J.: Process Mining from a Basis of State Regions. In: *Applications and Theory of Petri Nets (Petri Nets 2010)*. Volume 6128 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin (2010) 226–245
- [17] van der Aalst, W.M.P., Rubin, V., Verbeek, H.M.W., van Dongen, B.F., Kindler, E., Günther, C.W.: Process Mining: A Two-Step Approach to Balance Between Underfitting and Overfitting. *Software and Systems Modeling* **9**(1) (2010) 87–111
- [18] Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process Mining Based on Regions of Languages. In Alonso, G., Dadam, P., Rosemann, M., eds.: *International Conference on Business Process Management (BPM 2007)*. Volume 4714 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin (2007) 375–383
- [19] van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process Discovery using Integer Linear Programming. *Fundamenta Informaticae* **94** (2010) 387–412
- [20] Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering Block-structured Process Models from Incomplete Event Logs. In Ciardo, G., Kindler, E., eds.: *Applications and Theory of Petri Nets 2014*. Volume 8489 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin (2014) 91–110
- [21] Kryszkiewicz, M.: Fast Discovery of Representative Association Rules. In Polkowski, L., Skowron, A., eds.: *Rough Sets and Current Trends in Computing*. Volume 1424 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg (1998) 214–222
- [22] van der Aalst, W.M.P.: Decomposing Petri Nets for Process Mining: A Generic Approach. *Distributed and Parallel Databases* **31**(4) (2013) 471–507
- [23] Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM* **51**(1) (2008) 107–113

Business Process Measurement in small enterprises after the installation of an ERP software.

Stefano Siccardi and Claudia Sebastiani

CQ Creativiquadrati snc,
via Tadino 60, Milano, Italy
<http://www.creativiquadrati.it>

Abstract. We report the observation of the first six months of operation after the installation of an ERP software in a group of small Italian enterprises (some dealers of various products and one manufacturer). Before the ERP, no explicit process descriptions existed within the companies: the operations were manually performed, using office automation software or legacy programs that were not process oriented. The new ERP is equipped with a workflow engine, a number of standard processes that should be followed by the users, and a tracking system that logs the main steps of the processes. We use process mining tools to analyze the events logged by the ERP during the sales, the purchases and the manufacture cycles. Our aim is to 1) compare the ideal processes suggested by the ERP with the real paths followed by the users 2) describe the eventual adaptation of these paths, as the users became acquainted with the ERP 3) highlight critical segments in terms of time spent, iterations, etc. 4) compare the processes of different companies that are in similar business areas. The final goal is to get a better understanding of the processes and a rationalization of the operations. It must be stressed that both the ERP and the main tools used are open source, so that the process measurement is affordable even for very small (micro) enterprises.

Keywords: Business process mining, small enterprises, open source software, enterprise resource planning

1 Introduction

This report deals with the business process measurement in a sample of small enterprises in Italy. All the companies in the sample have recently adopted a new Enterprise Resource Planning (ERP) system, that is Odoo (previously OpenERP), available at www.odoo.com.

The introduction of the ERP was motivated by several reasons, mainly the necessity of replacing obsolete programs, the aim of integrating the software and the need to gain better control of operations. One of the main motivations of the choice of Odoo was its being an open source software.

The process control was not a primary goal at this stage: process design and measurement are usually activities too expensive, both in terms of time of the internal resources and of investment in consultancy, to be affordable for micro enterprises. So they were simply not considered by the entrepreneurs in the set of the realistic goals. On the other hand, processes in small enterprises are by no means simpler than in bigger ones, and a correct process understanding and design can be a major factor of success.

Another point that we were interested in was the companies' adaptation to the new ERP: although we customized the software in order to fit the users' business habits, it was unavoidable that they had somehow to change their ways of operating. So, in the first time of operation, we observed that they tried some different workflows before finding the one that best fit their needs. The changes in the processes during the first six months of operation, if any, can describe this adaptation phase.

As the Odoo system is intrinsically based on workflows, and automatically records many steps of the business process, it was evident that a very interesting byproduct of the ERP usage would have been the possibility to represent the processes as they are actually run. This can offer a number of interesting ideas to the companies' management.

Of course, the process representation would not have been possible without a process mining tool; we chose PROM rel. 6.4 (available at www.processmining.org), as it has all the capabilities we needed, and conforms to the open source philosophy of our projects. The basic ideas upon which PROM has been designed can be found e.g. in [1]; techniques to compute alignment between models and event logs are included in the software, and have been used in our analysis (see e.g. [2]).

For the preliminary data preparation we used Xesame ([3]).

The activity of process measurement aims to determine the performance of business processes, while the process monitoring is the activity to check them in order to discover some significant aspects (see e.g. [4]). In what follows, we concentrate on the first, and adopt a six months period of observation. There are also formal methods to study process adaptation and drift (see e.g. [5], [6]), however when considering the processes adaptation we limited ourselves to a qualitative approach.

1.1 The main process

We focused on the process of *selling - delivering goods or services - invoicing*, as it is the main one for all the companies of the sample. It is also a process that can be very complex, as customers can change their mind at several stages (before and after confirming an order, receiving goods, receiving invoices, etc.). Moreover it can be stopped and rescheduled several times if the goods to be delivered are not available, or if they arrive later at the companies' warehouses.

Three kinds of documents are involved in the process: a Sale Order (SO), containing the list of products and services that a customer wants to buy, with prices and payment terms; one or more delivery orders (OUT), generated by the

SO, with a list of products that must be taken from the warehouse and sent to the customer; one or more invoices (INV) that are fiscal documents specifying amounts to be paid with taxes and that can be created either from the SO or from the OUT(s).

When only services are sold no OUT is created, therefore the process is simpler.

We also consider the purchase process, consisting of the activity of ordering goods to suppliers and receiving them. Two documents are created: the Purchase Order (PO), with the list of products or services to buy; and one or more lists of items to receive at the warehouse (IN). Finally, for one company of the sample we also consider the manufacturing process, that is concerned with the Manufacture Order (MO), the list of products to produce. We limit ourselves to purchases and manufactures that are triggered by an SO, that is products are bought or manufactured just when a customer places an order for them.

The following events have been tracked for the documents described above:

SOcreate: a sales order is created by a user and is in the draft state. This is the starting point of the whole process.

SOmail: a sale order is sent to the customer

SOconfirm: a sale order is validated and reaches the confirmed status

OUTcreate: a list of goods to deliver is created in draft status

OUTconfirm: a list of goods to deliver has been checked and confirmed by the user

OUTwaiting: a list of goods to deliver is put in a waiting status, as something is not available

OUTready: a list of goods to deliver is ready to be sent

OUTsent: a list of goods to deliver has been sent to the customer

OUTinvoice: a list of goods to deliver has been invoiced

OUTcancel: a list of goods to deliver has been cancelled by the user

OUTbackord: a list of goods to deliver has been partially delivered and a back order has been created for the remaining items

OUTscrap: some items in list of goods to deliver have been marked as scrapped

OUTnot2invoice: a list of goods to deliver has been marked not to be invoiced (e.g. because it replaces something under warranty)

INVcreate: an invoice has been created in draft status

INVproforma: an invoice has been transformed in a pro-forma invoice

INVvalidate: an invoice has been confirmed by the user

INVcancel: an invoice has been cancelled

INVpay: an invoice has been paid

For the purchase process the events are:

POcreate: a purchase order is created in draft status

POconfirm: a purchase order has been confirmed by the user

INcreate: a list of goods to receive is created in draft status

INreceive: a list of goods has been received

INcancel: a list of goods to receive has been cancelled

For the manufacturing process the events are:

MOcreate: a manufacturing order is created in draft status

MOwaiting: a manufacturing order is waiting raw material

MOready: a manufacturing order is ready to produce

MOstart: a manufacturing order production started

MODone: a manufacturing order has been produced

All the above events are defined by the ERP systems, that automatically keeps track of their changes. The processes involving them have been mined and are described by the Petri nets in the next sessions.

The analysis method is the following:

1 - first we load in PROM the event log extracted by the Odoo database and preprocessed with Xesame

2 - we then take note of the main statistics of the complete log

3 - then we filter the cases in order to obtain cases that can be considered complete by a business point of view, e.g. cases that have been invoiced and paid, or, if not, that have been closed in some other reasonable way due to an agreement with the customer or else that can be analyzed as they have completed a definite section of the process

4 - next we build a model in Petri net format of the process as the company intends it

5 - at this stage we compare the filtered log to the model and draw considerations about its conformance and the general performance of the system

6 - we also analyze the filtered log to find users' behavioural patterns, e.g. if specific users are concerned with specific actions, or if some users tend to pass activities to others

7 - we again filter the log keeping only the cases started in the first month of operation, then those started in the last two months considered and repeat steps 5 and 6 on both sets

2 The cases

2.1 The sample companies

Our sample consists of six small Italian enterprises, that were observed during the first six months of their operations after the ERP implementation. We will use two digit codes to label them and we will group them according to their main business models:

1) Sale of goods that are kept in stock. Sample 01 (professional equipment and spare parts) and in part sample 06 (electromechanical devices) belong to this group.

2) Repair of customers' devices. Samples 02 and 03 belong to this group, both in the building industry, even if they sometimes sell devices and spare parts to their customers.

3) Sale of goods that are bought on the customer order and of maintenance services. Samples 04 and 05 belong to this group (electronic devices like PCs and telecommunication apparatus, software, etc.).

4) Manufacture to customer order. Sample 06 produces electromechanical components to fulfil customer orders, and sometimes to make stock, so in part belongs to this group and in part to group 1.

The event logs of sample enterprises are actually small, especially of samples 02 and 03; this is in part due to the fact that we chose the six months test period and the sample enterprises before knowing exactly how they would use the software and how many transactions they would process. Probably samples 02 and 03 are so small that it is even questionable if an ERP fits their needs.

However, as these kinds of enterprises are presently adopting ERP systems (as open source ERP systems are available), that forces them to follow structured processes, it is interesting to understand how they face this task.

2.2 General data

All the tracks of all the samples start with the SOcreate event, as expected. Some general data are summarized in table 1. The first lines refer to the total log as it is extracted from Odoo.

The second set of values refers to the log filtered keeping all the cases that end with INVpay, INVvalidate, OUTinvoice or OUTsent. This means that the process has been followed at least until some goods have been sent to the customer.

Table 1. General data of the samples

	01	02	03	04	05	06
Total Log						
Total tracks	2151	131	35	253	330	2105
Event classes	18	11	11	22	16	26
Events in the shortest track	1	1	2	1	1	1
Events in the longest track	61	20	12	64	50	66
Average events per track	11	9	8	9	9	13
Ending with INVpay	1064	73	11	45	125	845
Ending with INVvalidate	452	38	15	55	-	611
Ending with OUTsent	376	-	-	-	-	105
Log of completed cases						
Total tracks	1908	115	27	112	207	1579
Events in the shortest track	4	7	6	5	5	6
Events in the longest track	61	20	12	64	50	66
Average events per track	11	10	10	16	12	15
Min. event classes per track	4	10	6	4	5	6
Max event classes per track	14	11	11	17	15	19
Avg event classes per track	9	10	10	9	9	11

2.3 The 01 business sample

The process model is represented by the Petri net in figure 1. Sale orders always refer to products, even if some services are sold together, so at least one OUT is expected.

We note that:

- the first steps are related to the SO; when it is confirmed, the OUT is created
- when the OUT becomes ready, an invoice is created
- after invoice creation there may be a loop involving invoice cancelling and creating anew
- the process of sending goods may be repeated
- the last steps are validating the invoice and receiving payments
- no purchase orders are triggered by the sale orders, because the user sells goods that are kept in stock

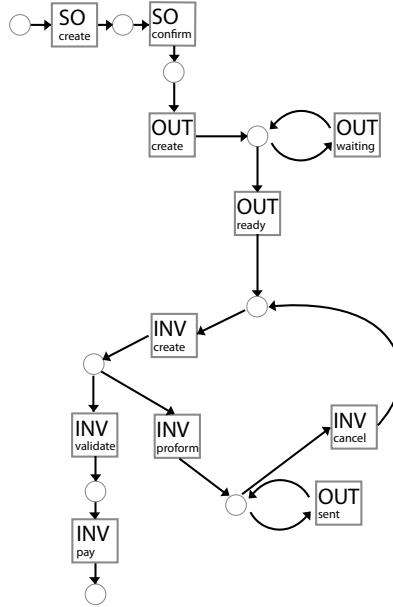


Fig. 1. The SO-OUT-INV process model for sample 01.

Applying the 1903 completed cases log onto this model, we found deviations mainly related to the events SOconfirm, OUTcreate, OUTwaiting, OUTready, INVcreate, INVvalidate, INVcancel and, to a lesser extent, INVproforma. The average throughput time is 1 month. From a time perspective, INVvalidate and INVcancel are critical; moreover, when a proforma invoice is issued, it is found a time lag between this event and the actual invoice. Applying only cases ending with INVpay (1064 items), we have similar results.

We then filtered the log for cases starting in the first month of the period, and found 244 tracks; and for events starting in the last two, and ending with one of the four events already considered by the end of the period, and found 178 tracks. These filtered logs, once applied onto the model, gave similar results. The only remarkable difference is the reduction in the average throughput time from 2 months to about 12 days. This is undoubtedly related to a better ability in the use of the new system and to the fact that in the first time of operation the users were still using their old software in parallel.

The PROM software can highlight actual cases that do not follow the model, this is useful to discuss with the users if the model has to be modified or if any unwanted actions are being performed. For instance, 255 cases follow the path:

SOcreate - SOconfirm - INVcreate - OUTinvoice - OUTsent - INVvalidate - INVpay

that is the invoice is created starting from the SO instead of the OUT. This is an alternative way of operating that can be included in the model.

On the other hand, 105 cases follow the path:

SOcreate - SOconfirm - OUTcreate - OUTwaiting - OUTwaiting - OUTready - INVcreate - OUTwaiting - OUTready - OUTsent - INVvalidate - INVpay

probably meaning that sale orders are modified later than expected, when an OUT has already been created. There are also much more complex examples, involving multiple sequences of OUTcreate - OUTconfirm - OUTcancel - OUTnot2invoice - OUTcreate etc.

This, in turn, can suggest the opportunity of clearer communication with the customers, so that they can place their orders correctly at the first time.

Finally, selective filtering of the log shows that all the users are involved in all the events, that is there is no user dedicated to a specific process section.

2.4 The 02 and 03 business samples

As 03 is a spin off of 02, these two samples are quite similar. Both the companies are very small and almost all the transactions were recorded by a single user, so a general uniformity of behaviour is found in the data.

The process model for both 02 and 03 is represented by the Petri net in figure 2. Sale orders refer always to products, even if some services are sold together, so at least one OUT is expected.

We note that in contrast with the 01 process, no loops are considered in this model.

Applying the log of completed cases of sample 02 (115 cases) and of sample 03 (27 cases) onto this model, we found deviations mainly related to the event INVcreate only. The average throughput time is 2.3 months for sample 02 and 1.35 for sample 03. From a time perspective, the main critical event for both samples is INVpay; this probably reflects a commercial issue and is not related to the process complexity. For sample 03 also OUTready is critical, whose average throughput time is 20 days; this probably reflects the fact that the company business mainly consists of fixing customers' devices, so when something arrives

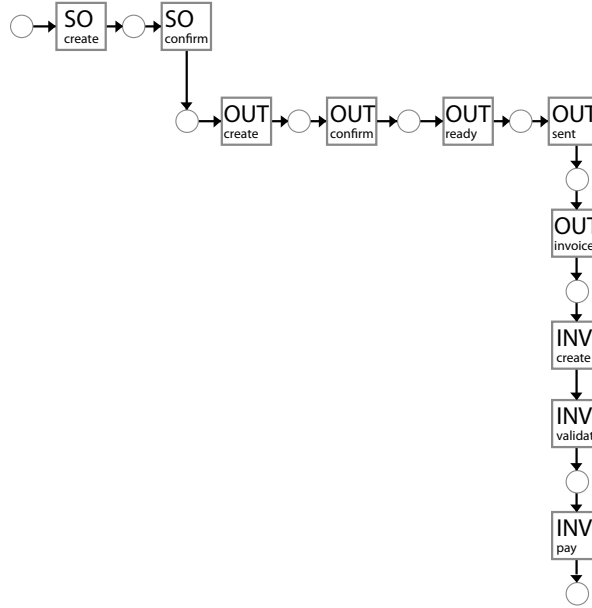


Fig. 2. The SO-OUT-INV process model for samples 02 and 03.

at their warehouse, time is needed to repair it before it can be sent back to the customer.

The filtered logs of sample 02 for cases starting in the first month and in the last two months, once applied onto the model, gave similar results.

As the cases are very few for sample 03 we did not compare the first months to the last ones.

Although the model is quite linear, the analysis of tracks highlighted some cases comprising loops and some degrees of complexity. Their number, however, was so low that they can be considered exceptional cases more than an indication of a more complex process. An example with multiple OUT and INV processing is shown in figure 3. Here an OUT can be cancelled after being confirmed and declared ready, and even when the invoicing process has started, by a manual action that has been represented as an "invisible" task (see e.g. [7]), so that it must be created again. Multiple invoices are created, one just after the SO confirmation, and others after sending goods.

2.5 The 04 and 05 business samples

In both samples, sales are of two different kinds: sometimes the user sells devices and equipment, that originate physical movement of goods (OUTs) and purchase orders; other times they sell services, e.g. assistance contracts, that do not imply any OUT. Moreover they manage assistance agreements with their customers, that have a single sale order and several invoices to be paid e.g. quarterly, so it

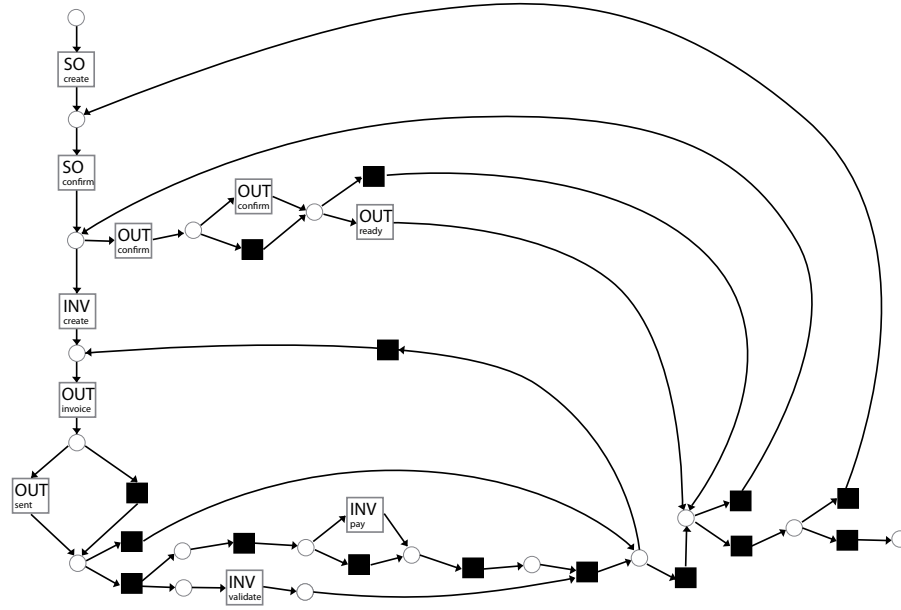


Fig. 3. A complex case of sample 02.

can happen that payment is not found as the last event in many recorded cases. The users usually buy products when a SO is received, so the PO process is triggered.

For both samples two business process models are found: for sample 04, 124 cases involve some physical products and 129 services only, for sample 05, 177 cases involve some physical products and 153 services only.

For 04 the filtered log of complete cases contains 58 cases involving products and 54 service cases.

For 05 the filtered log of complete cases contains 92 cases involving products and 115 service cases. Moreover 91 cases end with the SOmail event, that is they are quotations that the customers did not accept.

The first process is represented by the Petri net in figure 4 and the second in figure 5.

We note that for the first process:

- the first steps are related to the SO; when it is confirmed a PO is created
- when the PO is confirmed, an IN is created waiting for the purchased products; when products are received the OUT becomes ready
- after sending the OUT the invoice is created, validated and paid
- there can be change requests before invoice validation, that restart the process modifying the OUT

For the second process:

- the first steps are related to the SO; when it is confirmed an INV is created

- after invoice creation there is a loop involving invoice cancelling and creating anew, or receiving a payment and creating a new invoice

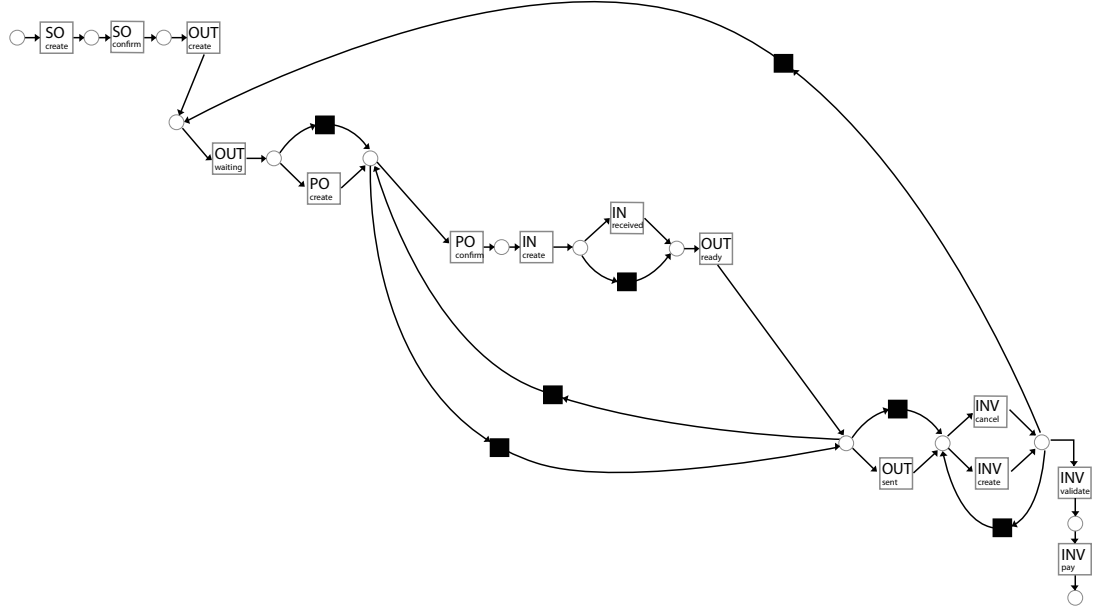


Fig. 4. The SO-PO-OUT-INV process model for products of samples 04 and 05.

Applying the cases log onto these models, we did not find significant deviations. For the product process of sample 04 the average throughput time is 1.4 months and for sample 05 it is 1.8 months. From a time perspective, no specific events are critical.

For the service process of sample 04 the average throughput time is 2.7 months. From a time perspective, INVcreate is critical; however, as these kind of sales involve long periods this could be a characteristic of the business (an invoice being issued quarterly in most cases).

For the service process of sample 05 the average throughput time is 1.7 months. From a time perspective, no specific events are critical. The logs are in good agreement with the model.

As the cases of sample 04 are very few, we did not compare the first month to the last ones.

For sample 05 comparing cases of first month to the last two, we found that in the first month the average throughput time was higher (2.45 months instead of 11 days for products and 2.5 months instead of 10 days for services).

Finally, selective filtering of the log shows that in sample 04 all the users are involved in all the events, moreover a single user was involved in about two thirds of the cases.

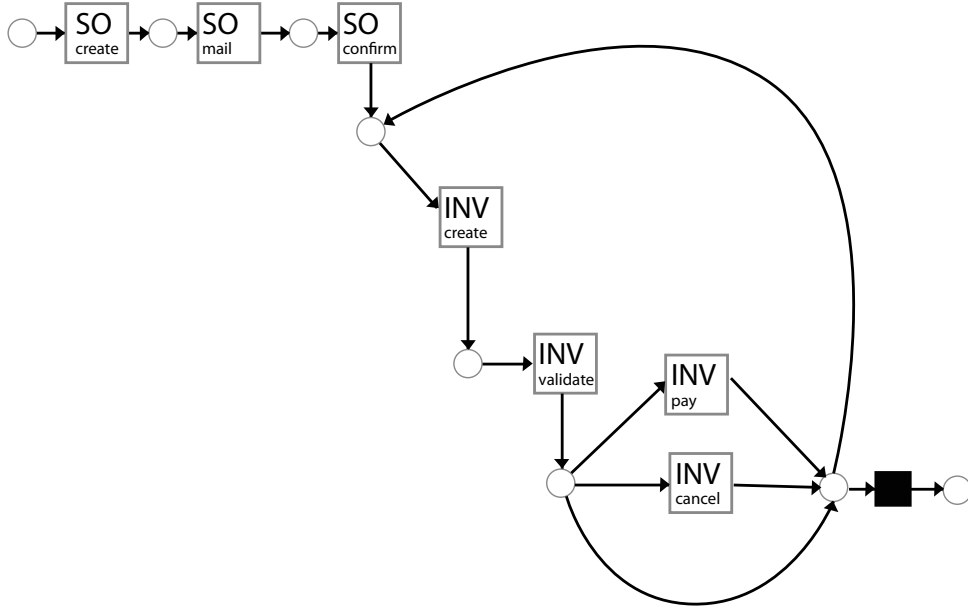


Fig. 5. The SO-INV process model for services of samples 04 and 05.

In sample 05 all the users are involved in the SO and OUT processing, but only 3 in invoicing and 2 in purchasing.

2.6 The 06 business sample

The user manufactures products both for keeping stock and to fulfil specific sale orders, so the MO process is triggered by an SO for some products. Most sales involve products, although 179 service SO have been found.

The filtered log of complete cases contains 441 cases involving products specifically manufactured for the SO and 1138 cases involving products taken from the warehouse.

The first process is represented by the Petri net in figure 6, while the second is similar to the one already analyzed in figure 1.

We note that for the first process:

- the first steps are related to the SO; when it is confirmed a MO is created
- when the MO is done, the OUT becomes ready
- after sending the OUT the invoice follows its usual process

Applying the cases log onto these models, we did not find significant deviations. For the first process the average throughput time is 3.1 months. This long time is probably due to customers placing their orders even months in advance. From a time perspective, the only event that needs further analysis is the MOREady as it is sometimes delayed for lack of raw materials, or sometimes due to an agreement with the customer about the delivery date. The main deviations from

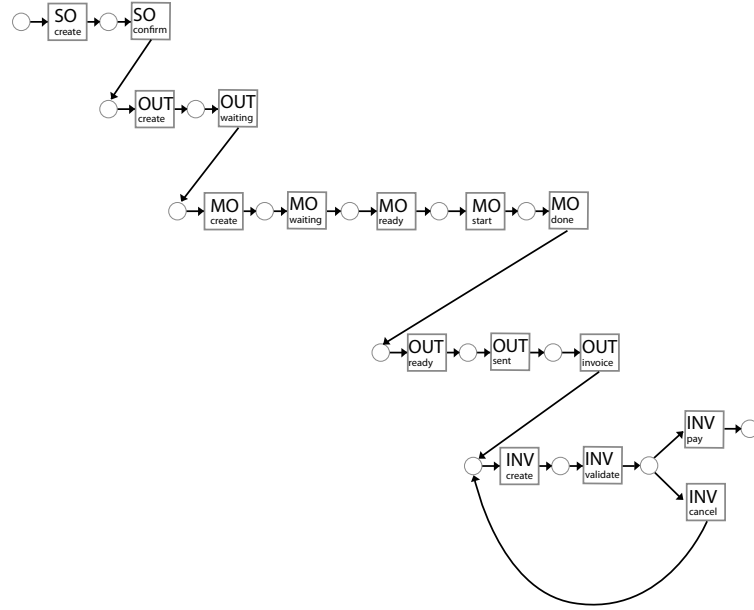


Fig. 6. The SO-MO-OUT-INV process model for sample 06.

the model are cases where products are delivered in two or more shipments, and backorders are created.

For the second process the average throughput time is 2.5 months. From a time perspective, no specific events are critical. Some deviations from the model are found for the events SOconfirm and OUTconfirm, probably due to late changes to the SO requested by the customers.

Comparing the first month of operation with the last two, we found that the main difference is a decrease in throughput time and in case length, that probably means that the users gradually become acquainted with the system and do not need to cancel and redo operations if not in exceptional cases.

Finally, selective filtering of the log shows that although 9 users recorded some transactions, sales orders and invoices were usually managed by 3 users, manufacturing by 2 others and delivering by all of them.

3 Discussion

The results of our analysis show that, in general, the processes follow the models that the users have in mind for their "normal" business cases. This is somewhat in contrast with the subjective impression that the users themselves have of their own business.

In the preliminary meetings that we had, to check if the ERP would actually fit the companies' needs, the users described many complex situations that could

arise, as if they were very common. Looking at the data, it turned out that these are actually exceptions; it is true that they can be very involved, and can generate time loss and dissatisfaction, but they are rarer than expected.

Although the data are too few to get exact statistics, the indication is that criticality is related to exceptional cases and not to the average ones.

We discussed specific cases, like that of fig. 3, with the users, and found that the main source of complexity, for all samples, consists of the customers' requests to change their agreements at a late stage. That is, some customers want to change their SO after confirming them, or even after they have received the products; or they want to change the shipment or payment conditions, etc. This means that documents must be cancelled and processed again, often with consequences on other documents.

It is interesting that we found two different reactions to the above remarks. Some managers told us that their goal is to be flexible, and that to fulfill their customers requests is exactly their mission. They think that complexity is inherent to their job, and used the cases logs to ask new implementations to our ERP to fit the way they run their business.

Other managers, on the contrary, realized that simpler processes are more efficient, and decided to follow the ERP standard processes for as long as possible. They defined some new internal rules for this, e.g. a standard time to wait before confirming the main documents (e.g. SO). If a customer wants to change something after, he will have to pay an extra fee for the reprocessing of the order.

4 Conclusions

We applied process mining techniques to a sample of micro enterprises, that would not be able to consider their own processes in other ways. The main goal of this activity is, in our opinion, to bring process design to the managers' attention.

About our business sample, we observe that:

- processes of similar business are very similar, for instance those of sample 04 and 05, and in part also of sample 01 and 06
- in general the actual processes follow the models, and specific deviations can be discussed with the managers to suggest new ideas; anyway they are more often exceptional cases than variants of the models
- we did not observe process modification or adaptation in time; the main difference between the first and last months is an increase in performance. As it is not realistic that the business itself has quickened so much in six months, this is probably due to an increase in the users' familiarity with the system
- especially in the smaller samples, there are no fixed correlations between tasks and people (everybody does everything), while bigger companies tend to assign specific roles to the users

Finally, we note that in this analysis we used the standard events traced by Odoo, but it is possible to trace any events that the users think could be relevant for their business.

References

1. Weijters, A.J.M.M., van der Aalst, W.M.P.: Workflow Mining: Discovering Workflow Models from Event-Based Data, in C. Dousson, F. Happner, and R. Quiniou, editors, *Proceedings of the ECAI Workshop on Knowledge Discovery and Spatial Data*, 78–84, (2002)
2. Adriansyah, A.: Aligning observed and modeled behavior, PhD Thesis. Technische Universiteit Eindhoven, Eindhoven, The Netherlands, (2014)
3. Buijs, J.C.A.M.: Mapping Data Sources to XES in a Generic Way, Master Thesis, Technische Uviversiteit Eindhoven, Eindhoven, The Netherlands, (2010)
4. Kueng, P., Hagen, C., Rodel, M., Seifert, S.: Business Process Monitoring and Measurement in a Large Bank: Challenges and selected Approaches, In: *Proceedings of the 16th International Workshop on Database and Expert Systems Applications (DEXA2005)*, IEEE Press, New York, 955–961, (2005)
5. Hallerbach, A., Bauer, T., Reichert, M.: Capturing Variability in Business Process Models: The Provop Approach, *Journal of Software Maintenance and Evolution: Research and Practice*, 519–546, (2010)
6. Bose, R.P.J.C., van der Aalst, W.M.P., Zliobaite, I., Pechenizkiy, M.: Dealing With Concept Drifts in Process Mining, *IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS*, (2014)
7. Wen, L., Wang, J., Van Der Aalst, W.M.P., Huang, B., Sun, J., Mining process models with prime invisible tasks, *Data and Knowledge Engineering*, vol. 69, 999–1021, (2010).

Reasoning on Data-Aware Business Processes with Constraint Logic

Maurizio Proietti and Fabrizio Smith

National Research Council, IASI "Antonio Ruberti" - Via dei Taurini 19, 00185 Roma, Italy
{maurizio.proietti, fabrizio.smith}@iasi.cnr.it

Abstract. We propose a framework grounded in Constraint Logic Programming for representing and reasoning about business processes from both the workflow and data perspective. In particular, our goal is twofold: (1) define a logical language and a formal semantics for process models where data object manipulation and interactions with an underlying database are explicitly represented; (2) provide an effective inference mechanism that supports the combination of reasoning services dealing with process behavior and data properties. To this end we define a rule-based process representation coping with a relevant fragment of the popular BPMN modeling notation, extended with annotations that model data manipulation. The behavioral semantics of a process is defined as a state transition system by following an approach similar to the Fluent Calculus, and allows us to specify state change in terms of preconditions and effects of the enactment of activities. Our framework provides a wide range of reasoning services, which can be performed by using standard Constraint Logic Programming inference engines.

Keywords: Business Process, Constraints, Logic Programming, Analysis, Verification

1 Introduction

The penetration of Business Process (BP) Management solutions into production realities is constantly growing, due to its potential for an effective support to enterprise actors and business stakeholders along the entire BP life-cycle. In this frame, modeling languages such as BPMN [1] are largely adopted by the stakeholders (designers, analysts, business men) to develop conceptual models to be used for the design and reengineering of BPs. One of the main advantages of having a machine-processable representation of BPs available is that it enables the automation of tasks dealing with process analysis, simulation and verification.

However, standard process-centric approaches focus on the procedural representation of a BP as a workflow graph that specifies the planned order of operations, while the interactions of individual operations with the underlying *data* layer is often left implicit or abstracted away. Indeed, the automated analysis issue is addressed in the workflow community mainly from a control flow perspective (see, for instance, the notion of soundness [2]), and most of the tools today available aim at verifying whether the behavior of the modeled system enforces requirements specified without considering the data perspective.

In order to provide an integrated account of the workflow and data modeling, several approaches have been proposed both in industrial realities (e.g., [3, 4]), as well as in the database (e.g., [5]) and workflow (e.g., [6]) research communities. A *data-aware* BP representation explicitly models the manipulation of data objects operated by individual tasks and their interactions with databases, with the aim of enabling the automated analysis of behavioral properties of the resulting system.

In this paper we propose a logic-based framework for representing and reasoning about data-aware BP models, where the workflow perspective, specified according to BPMN, is enriched by annotations defining *preconditions* and *effects* of individual process elements in terms of *data objects*, used to store information that is read and modified by the process enactment. We also consider the existence of an underlying database, which can be queried during the enactment, retrieving values to be used for data object manipulation. The behavioral semantics of a process is defined as a state transition system by following an approach derived from the *Fluent Calculus* [7], and by integrating into the framework a *symbolic* representation of data object values, given in terms of arithmetic constraints over the real numbers. The proposed rule-based formalization supports a relevant fragment of BPMN in addition to expressive data modeling, and its grounding into Constraint Logic Programming (CLP) [8] provides a uniform and formal framework that enables automated reasoning.

In this work we do *not* propose yet another business process modeling language, but we assume a pragmatic perspective aiming at supporting process-related knowledge expressed by means of de-facto standards for BP modeling, like BPMN. To the best of our knowledge, this is one of the first attempts to provide a formal execution semantics for expressive BPMN workflows in the presence of data and arithmetic constraints. Notably, the CLP formalization directly provides an executable semantics, which enables the implementation of analysis and verification tasks relying on established automated reasoning methods and tools. Due to the presence of data, the state space of the modeled systems is potentially infinite and most verification problems are undecidable. However, the symbolic representation of data values by means of constraints achieves the termination of a number of reasoning services in many cases of practical relevance also in the presence of an infinite state space.

The paper is organized as follows. After presenting a motivating scenario and introducing the modeling framework in Section 2, we provide a formal account of the behavioral semantics in Section 3. In Section 4 we show how automated reasoning methods developed in the field of CLP can be directly applied to perform analysis and verification tasks. In Section 5 we present a proof-of-concept prototype, and, finally, in the concluding section we give a critical discussion of our approach, along with directions for future work.

2 Modeling Data-Aware Business Processes

A data-aware Business Process Schema (*DAPS*) is an activity workflow model, where each task can additionally operate on *data objects* used to store information that is read and modified by process enactment. In our approach, data objects are essentially regarded as variables, and hence at any time during execution there is a single instance

of a given data object that may be read or (over-)written by some activity. We consider two main types of relationships between activities and data objects. Firstly, the enactment of an activity may be guarded by a condition involving a number of data objects. Secondly, the enactment of an activity can modify the value of a data object, hence producing a new value, possibly related to other data objects' values by an *arithmetic constraint* over the real numbers \mathbb{R} . We also consider the presence of a database (*DB*) that can be queried during process enactment, hence retrieving values to be used for data object manipulation. For the scope of this work, we assume that *DB* cannot be updated by activity executions. Furthermore, we assume to deal with BPs whose state space, considering the control flow only, is finite. However, since the data objects can assume infinitely many values, the state space of the overall system is in general infinite.

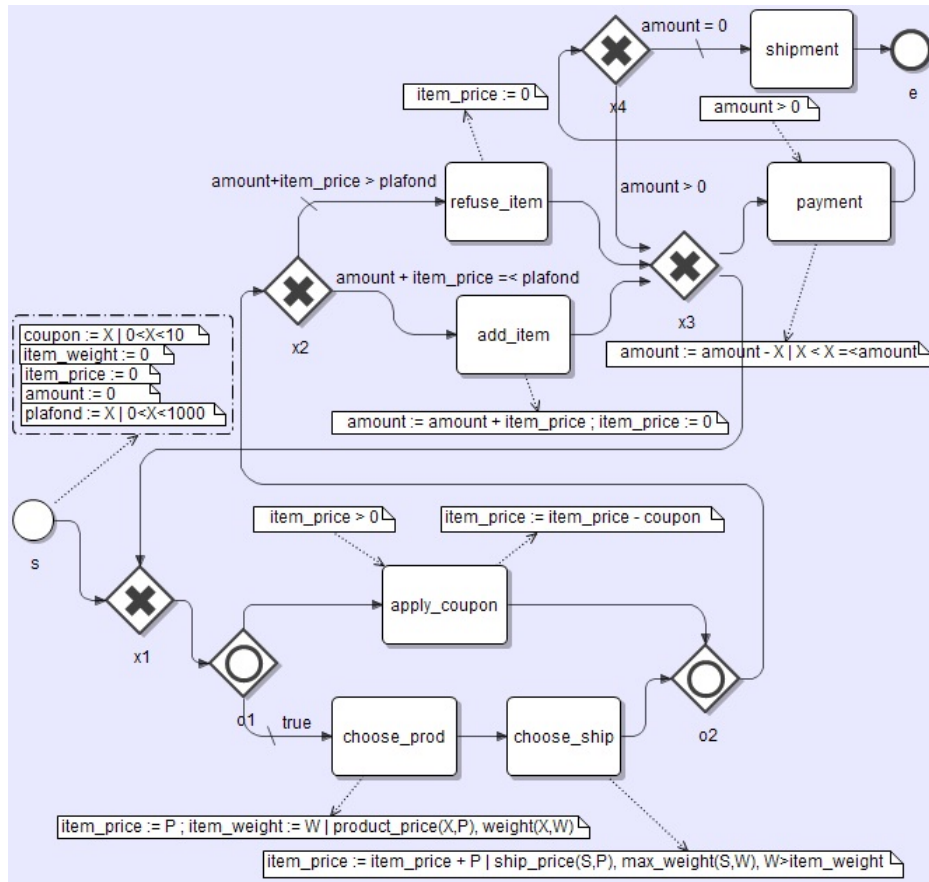


Fig. 1: Example DAPS for an eProcurement Scenario

For the representation of the workflow-related perspective, we mainly refer to the BPMN standard [1], which will be used throughout as a reference notation. We show how a DAPS is specified by means of the example depicted in Figure 1, which deals with the handling of a purchase order in an eProcurement scenario.

A DAPS consists of a set of *flow elements* and *relations* between them, and it is associated with a unique *start event* (e.g., s) and a unique *end event* (e.g., e), which are flow elements that represent the entry point and the exit point, respectively, of the process. An *activity* is a flow element that represents a unit of work performed within the process. It can be modeled as a *task*, representing an atomic activity no further decomposable (e.g., *choose_prod*), or as a *compound activity*, representing the execution of a sub-processes (not exemplified here). The sequencing of flow elements is specified by the *sequence flow* relation (corresponding to solid arrows), and the *branching/merging* of the control flow is specified by using three types of *gateways*: *exclusive* (XOR, e.g., $x1$), *inclusive* (OR, e.g., $o1$), and *parallel* (AND, not exemplified here).

The *item flow* relation (corresponding to dotted arrows) specifies that a flow element uses as *input* or manipulates as *output* particular data objects. In our setting, the input and output of a flow element can be enriched by declarative descriptions of *preconditions* and *effects*, respectively, formulated in terms of arithmetic constraints, value assignments (denoted by $:=$), and database *queries* (following the $|$ symbol). For instance, the task *apply_coupon* requires (precondition) a value of *item_price* greater than 0, while upon its execution (effect), the value of *item_price* is decreased by the amount of *coupon*. More complex effects can be specified, as in the case of *choose_prod*, where the values assigned to *item_price* and *item_weight* are retrieved by performing the conjunctive database query $product_price(X, P), weight(X, W)$. Uppercase letters denote *variables*, representing values not known at design-time that are introduced during the enactment, e.g., retrieved by database queries or produced after interactions with external systems or users. The values that variables are allowed to assume at run-time can be characterized in terms of arithmetic constraints, as exemplified by the *payment* effect where, due to the constraint $0 < X \leq amount$, the possible values of X after each execution of *payment* range from 0 to the current value of *amount* (representing any admissible paid amount). Similarly to activity preconditions, *guards* can be attached to outgoing sequence flows of inclusive and exclusive gateways, as in the case of $o1$ and $x2$. Whenever a guard is not defined, a non-deterministic behavior is assumed.

The depicted BP is started by the user log-in, which triggers the execution of the start event leading to the initialization of the involved data objects. The user can select a number of products, by choosing for each of them a shipment compatible with the item weight and optionally applying a coupon, which decreases the item price. When the amount due exceeds the plafond associated with the user, a selected product cannot be added to the order and it is refused. In order to proceed with the shipment, the full amount due has to be paid, possibly through several subsequent payments.

3 Formal Semantics

Now we present a formal definition of the behavioral semantics, or *enactment*, of a DAPS, by following an approach derived from the *Fluent Calculus*, a well-known rule-based calculus for action and change (see [7] for an introduction), which is formalized in Logic Programming (LP). In [9] we have proposed a specialized version of the Fluent Calculus, developed to specifically deal with BPs. Here, in order to cope with the data perspective, our formalization is enhanced by using Constraint Logic Programming

(CLP) [8], which extends LP with constraints over specific domains and, in particular, over the domain of the real numbers with the usual arithmetic operations.

3.1 Constraint Logic Programming

We will consider CLP programs with constraints over the set \mathbb{R} of the real numbers. We will denote variables by upper case letters X_1, X_2, \dots , while we will denote constants, predicate and function symbols by lower case letters, a, p, f, \dots . Constraints are inductively defined as follows. An *atomic constraint* is either a formula of the form $p_1 \geq p_2$ or a formula of the form $p_1 > p_2$, where p_1 and p_2 are polynomials with real variables. We will also use the equality ‘=’ and the inequalities ‘ \leq ’ and ‘ $<$ ’ defined in terms of ‘ \geq ’ and ‘ $>$ ’ as usual. A *constraint* is either *true*, or *false*, or an atomic constraint, or a *conjunction* of constraints.

An *atom* is an atomic formula of the form $p(t_1, \dots, t_m)$, where p is a predicate symbol not in $\{\geq, >\}$ and t_1, \dots, t_m , with $m \geq 0$, are terms. A *literal* is either an atom A or a negated atom $\neg A$. A *goal* is a (possibly empty) conjunction of atoms. A *constrained goal* $c \wedge G$ is a conjunction of a constraint c and a goal G . A CLP *program* is a finite set of *rules* of the form $A \leftarrow c \wedge G$ (to be understood as “ A if c and G ”), where A is an atom and $c \wedge G$ is a constrained goal. Given a rule $A \leftarrow c \wedge G$, A is the *head* of the rule and $c \wedge G$ is the *body* of the rule. A rule with empty body is called a *fact*. A term or a formula is *ground* if no variable occurs in it.

Let $T_{\mathbb{R}}$ denote the set of ground terms built from \mathbb{R} and from the set of function symbols in the language. An \mathbb{R} -*interpretation* is an interpretation that: (i) has universe $T_{\mathbb{R}}$, (ii) assigns to $+$, \times , $>$, \geq the usual meaning in \mathbb{R} , and (iii) is the standard *Herbrand interpretation* [10] for function and predicate symbols different from $+$, \times , $>$, \geq . We can identify an \mathbb{R} -interpretation I with the set of ground atoms (with arguments in $T_{\mathbb{R}}$) which are true in I . We write $\mathbb{R} \models \phi$ if ϕ is true in every \mathbb{R} -interpretation. A constraint c is *satisfiable* if $\mathbb{R} \models \exists X_1, \dots, X_n. c$, where X_1, \dots, X_n are all variables occurring in c . A constraint c *entails* a constraint d , denoted $c \sqsubseteq d$, if $\mathbb{R} \models \forall X_1, \dots, X_n. c \rightarrow d$, where X_1, \dots, X_n are all variables occurring in c or d . An \mathbb{R} -*model* of a CLP program P is an \mathbb{R} -interpretation that makes true every rule of P . Every CLP program P has a *least* (with respect to set inclusion) \mathbb{R} -model, denoted $M(P)$.

We consider the standard operational semantics of CLP programs based on *resolution* extended with *constraint solving* [8]. For reasons of simplicity we assume that no negated atom appears in the body of a CLP rule. However, we consider negated atoms in queries. A *query* has the form $\leftarrow c \wedge C$, where c is a constraint and C is a conjunction of literals. A query $\leftarrow c \wedge C$ *succeeds* if it is possible to derive from it, possibly in many steps, a query of the form $\leftarrow c' \wedge \text{true}$, where c' is a satisfiable constraint and *true* is the empty conjunction of literals. The constraint c' is also called an *answer* to the query $\leftarrow c \wedge C$. As usual in CLP systems, we assume the left-to-right selection strategy of literals during resolution. A query $\leftarrow c \wedge C$ (*finitely*) *fails* if the set of derivations from it is finite and no query of the form $\leftarrow c' \wedge \text{true}$, where c' is satisfiable, is derivable. The operational semantics is sound with respect to the least model semantics in the sense that, if a query $\leftarrow Q$ succeeds with answer c' then $M(P) \models \forall X_1, \dots, X_n. c' \rightarrow Q$, where X_1, \dots, X_n are the variables occurring in $c' \rightarrow Q$. If $\leftarrow Q$ fails then $M(P) \models \forall X_1, \dots, X_n. \neg Q$, where X_1, \dots, X_n are the variables occurring in Q .

3.2 Data-Aware BP Schema Representation

A DAPS is formally represented by a triple $\langle WF, DC, DBS \rangle$, where WF is a workflow specification, DC (*data constraints*) is a specification of the *preconditions* and *effects* of activities on data objects, and DBS is a database schema.

The workflow specification WF is a set of ground facts of the form $p(a_1, \dots, a_n)$, where a_1, \dots, a_n are constants denoting individual flow elements and p is a predicate symbol representing a BPMN construct (e.g., activity, gateway, sequence flow). For instance, the formal specification of the eProcurement BP in Figure 1 will contain facts such as: $bp(eProc, s, e)$, stating that $eProc$ is a process starting with s and ending with e ; $task(choose_prod)$, stating that $choose_prod$ is an atomic activity within the workflow; $exc_branch(x2)$, stating that $x2$ is an exclusive branch point, i.e., a decision point; $seq(choose_prod, choose_ship, eProc)$, stating that a sequence flow relation is defined between $choose_prod$ and $choose_ship$ in $eProc$.

The data constraints DC specify the way data objects are manipulated during process enactment, by means of the following relations:

Precondition: $pre(A, C, Proc)$, which specifies an *enabling condition* C that the data objects must satisfy to enable the execution of an activity A in the process $Proc$;

Effect: $eff(A, E, Proc)$, which specifies the *effect expression* E describing the effect on the data objects of executing A in the process $Proc$;

Guard: $guard(C, B, Y, Proc)$, which specifies a *conditional sequence flow* used to select the set of successors of decision points, where the *enabling condition* C must hold in order to enable the flow element Y , successor of B in the process $Proc$.

Enabling conditions and effect expressions are formally defined as follows. Let DO denote the set of data objects occurring in the DAPS. An *arithmetic expression* is an expression constructed from DO , (CLP) variables, real numbers, $+$, and \times . A *constraint expression* is an expression of the form $a_1 \text{ rel } a_2$, where a_1, a_2 are arithmetic expressions and $\text{rel} \in \{\geq, >\}$ (i.e., a constraint expression is an atomic constraint on data objects and CLP variables). A *db-query* is an atom whose predicate is defined in the database schema DBS . A *data update* is an expression of the form $o := a$, where $o \in DO$ and a is an arithmetic expression. An *enabling condition* is the conjunction of $n \geq 0$ constraint expressions. An *update condition* is either a constraint expression or a db-query. An *effect expression* is a pair $\text{data-updates} \parallel \text{conds}$, where *data-updates* is a sequence of data updates and *conds* is a conjunction of update conditions.

Returning to the eProcurement example of Section 2, a precondition associated with the *payment* activity is:

$$pre(payment, [amount > 0], eProc)$$

meaning that the task *payment* can be executed only if the data object *amount* has a positive value. In the above example and in the sequel, both sequences and conjunctions appearing as predicate arguments are represented using lists. The specification of the effect associated with the *choose_ship* activity is:

$$eff(choose_ship, [item_price := item_price + P] \parallel [ship_price(S, P), max_weight(S, W), W > item_weight], eProc)$$

meaning that the effect of the execution of the task *choose_ship* is that the value of the data object *item_price* is incremented by a quantity P , where P is the price of a shipment type S and the value of *item_weight* is below the maximum weight allowed for S .

The specification of the guard associated with the flow from $x2$ to *refuse_item* is:

$guard([amount + item_price > plafond], x2, refuse_item, eProc)$

meaning that the control flow can proceed from the gateway $x2$ to the *refuse_item* task only if the sum of the values of *amount* and *item_price* exceeds the value of *plafond*.

The database schema *DBS* consists of a set of predicate symbols (with arity) representing the relations stored in the database, together with a set of formulas of the form $p(X_1, \dots, X_n) \rightarrow c$, where p is a predicate symbol in the schema and c is a constraint whose variables are among X_1, \dots, X_n , representing *integrity constraints* (for simplicity we do not consider more complex integrity constraints). A *database instance* of *DBS* is a finite set of ground facts $p(a_1, \dots, a_n)$ that satisfies all formulas in *DBS*.

3.3 Behavioral Semantics of Data-Aware Processes

Similarly to the Fluent Calculus, we represent the state of the world as a set of *fluents*, i.e., terms denoting atomic properties that hold at a given instant of time. The execution of a flow element may cause a change of state, i.e., an update of the collection of fluents associated with it. In particular, a change of state can be determined by the effects of activities specified by *DC*. A fluent is represented by an expression of the form $f(a_1, \dots, a_n)$, where f is a fluent symbol and a_1, \dots, a_n are constants or variables. We take a closed-world interpretation of states, that is, we assume that a fluent F holds in a state S iff $F \in S$. Our set-based representation of states relies on the assumption that the DAPS is *safe*, that is, during its enactment there are no concurrent executions of the same flow element [2]. This assumption enforces that, in absence of data objects, the set of states reachable by a given DAPS is finite.

We will consider the following three kinds of fluents:

- $cf(E_1, E_2, Proc)$, which means that the flow element E_1 has been executed and the successor flow element E_2 is waiting for execution, during the enactment of the process *Proc* (*cf* stands for *control flow*);
- $en(A, Proc)$, which means that the activity A is being executed during the enactment of the process *Proc* (*en* stands for *enacting*).
- $val(O, V)$, which means that the data object $O \in DO$ has value V .

The truth value of a fluent or update condition depends on the state where it is evaluated. For this reason we introduce the satisfaction relation $holds(C, S)$, which holds if C is true in the state S , where C is either a fluent or a conjunction of update conditions.

1. $holds([], S)$
2. $holds([C|Cs], S) \leftarrow holds(C, S) \wedge holds(Cs, S)$
3. $holds(p(X_1, \dots, X_n), S) \leftarrow p(X_1, \dots, X_n)$ for every p declared in *DBS*
4. $holds(F, S) \leftarrow fluent(F) \wedge F \in S$
5. $holds(val(X, V), S) \leftarrow constant(X) \wedge V = X$
6. $holds(val(X, V), S) \leftarrow variable(X) \wedge V = X$
7. $holds(A_1 > A_2, S) \leftarrow V_1 > V_2 \wedge holds(val(A_1, V_1), S) \wedge holds(val(A_2, V_2), S)$
8. $holds(val(A_1 + A_2, V), S) \leftarrow V = V_1 + V_2 \wedge holds(val(A_1, V_1), S) \wedge holds(val(A_2, V_2), S)$

Recall that in this paper we assume that the database does not change during process enactment, and hence in Rule 3 the evaluation of the db-query $p(X_1, \dots, X_n)$ does not depend on the state S . Rule 4 has one instance for each kind of fluents, and a particular instance is the following rule for retrieving the value of a data object O in a state S : $holds(val(O, V), S) \leftarrow val(O, V) \in S$. Rules 5 and 6 express the fact that the value of logical variables and constants is independent of the state. Rules 7 and 8 are needed to define the value of constraint expressions by structural induction. We have omitted the rules for \geq and \times , which are similar to rules 7 and 8, respectively.

The change of state determined by the execution of an action will be formalized by a relation $result(S_1, A, S_2)$, which holds if action A can be executed in state S_1 leading to state S_2 . For the definition of $result(S_1, A, S_2)$, we assume that an instance of the database schema DBS is provided. We also assume that the execution of an activity has a beginning and a completion (although we do not associate a *duration* with activity execution), while the other flow elements execute instantaneously. Thus, we will consider two kinds of actions: $begin(A)$ which starts the execution of an activity A , and $complete(E)$, which represents the completion of the execution of a flow element E (possibly, an activity). The following auxiliary predicate will be used: $update(S_1, T, U, S_2)$, which holds if $S_2 = (S_1 - T) \cup U$, where S_1, T, U , and S_2 are sets of fluents.

Let us now present some of the rules that define the behavioral semantics of a DAPS. The state change determined by the execution of a task is defined by the following two rules, corresponding to the start and the completion of the task, respectively:

$$\begin{aligned} result(S_1, begin(A), S_2) &\leftarrow task(A) \wedge holds(cf(X, A, P), S_1) \wedge pre(A, C, P) \wedge holds(C, S_1) \\ &\quad \wedge update(S_1, \{cf(X, A, P)\}, \{en(A, P)\}, S_2) \\ result(S_1, complete(A), S_2) &\leftarrow task(A) \wedge holds(en(A, P), S_1) \wedge seq(A, Y, P) \\ &\quad \wedge eff(A, DU \parallel C, P) \wedge holds(C, S) \wedge apply(DU, S_1, S') \\ &\quad \wedge update(S', \{en(A, P)\}, \{cf(A, Y, P)\}, S_2) \end{aligned}$$

The first rule states that the execution of task A is started if the control flow has reached it ($holds(cf(X, A, P), S_1)$) and the enabling conditions associated with it hold in the current state ($pre(A, C, P) \wedge holds(C, S_1)$). The successor state is obtained by asserting that the process is enacting A ($update(S_1, \{cf(X, A, P)\}, \{en(A, P)\}, S_2)$). The second rule states that the execution of task A can be completed if the update conditions associated with A hold in the current state ($eff(A, DU \parallel C, P) \wedge holds(C, S)$). The successor state is obtained by applying the sequence DU of data updates, hence updating the values of the data objects ($apply(DU, S_1, S')$), and moving the control flow to the next flow element Y ($update(S', \{en(A, P)\}, \{cf(A, Y, P)\}, S_2)$). The relation $apply(DU, S_1, S')$, meaning that state S' is obtained from state S_1 by performing the sequence DU of data updates is defined as follows:

$$\begin{aligned} apply([], S, S) & \\ apply([DU|DUs], S, T) &\leftarrow apply(DU, S, S') \wedge apply(DUs, S', T) \\ apply(O := A, S, S') &\leftarrow holds(val(A, V), S) \wedge update(S, \{val(O, X)\}, \{val(O, V)\}, S') \end{aligned}$$

The following two rules formalize the state changes determined by the execution of an exclusive branch (with a guard associated with an outgoing flow) and an exclusive merge, respectively.

$$\begin{aligned} result(S_1, complete(B), S_2) &\leftarrow exc_branch(B) \wedge holds(cf(X, B, P), S_1) \wedge seq(B, Y, P) \\ &\quad \wedge guard(C, B, Y, P) \wedge holds(C, S_1) \wedge update(S_1, \{cf(X, B, P)\}, \{cf(B, Y, P)\}, S_2) \end{aligned}$$

$$\begin{aligned} result(S_1, complete(M), S_2) \leftarrow & exc_merge(M) \wedge holds(cf(A, M, P), S_1) \wedge seq(M, Y, P) \\ & \wedge update(S_1, \{cf(A, M, P)\}, \{cf(M, Y, P)\}, S_2) \end{aligned}$$

Note that, in particular, in order to proceed from the exclusive branch B to the next flow element Y , the guard C associated with the flow from B to Y should hold in the current state ($guard(C, B, Y, P) \wedge holds(C, S_1)$).

The behavioral semantics of other flow elements, e.g., parallel or inclusive gateways, can be formalized by rules defined in a similar style. For lack of space we omit those rules and we refer to [9] for more details in the simpler case where data object manipulation (which is the main contribution of this paper) is not considered¹.

The relation $r(S_1, S_2)$ holds if a state S_2 is *immediately reachable* from a state S_1 , that is, some action A can be executed in state S_1 leading to state S_2 :

$$r(S_1, S_2) \leftarrow result(S_1, A, S_2)$$

We say that a state S_2 is *reachable* from a state S_1 if there is a finite, possibly empty, sequence of actions from S_1 to S_2 , that is, $reachable_state(S_1, S_2)$ holds, where the relation $reachable_state$ is the reflexive-transitive closure of r .

4 Reasoning Services

The formal semantics of data-aware BP schemas introduced in Section 3 is the basis for developing automated reasoning techniques for the analysis and verification of business processes that manipulate data objects. A major point is that our formal semantics is a CLP program, and hence we can directly apply automated reasoning methods and tools developed in the field of Constrained Logic Programming to perform analysis and verification tasks. Indeed, by using standard CLP systems we are able to provide a framework that supports several reasoning services and, in particular, in this section we will demonstrate some of them and their use for analyzing process *enactment*, for *testing* process executions, and for *verifying* behavioral properties.

Given a DAPS $\langle WF, DC, DBS \rangle$ and a database instance D of the schema DBS , let \mathcal{KB} be the CLP program consisting of: (1) the ground facts WF specifying the workflow, (2) the *pre*, *eff*, and *guard* facts in DC specifying the enabling conditions, effects, and guards associated with the workflow, (3) the ground facts in D , and (4) the rules (introduced in Section 3) that define the behavioral semantics of the DAPS.

Reasoning services will be realized by evaluating queries to the CLP program \mathcal{KB} . One major advantage of our approach is that query evaluation relies on a *symbolic* representation of states, which often avoids the actual exploration of the whole, in general infinite, state space, by covering that space by means of a finite set of constraints. More specifically, a symbolic state is represented as a set of the form:

$$\{f_1, \dots, f_k, val(o_1, V_1), \dots, val(o_m, V_m)\} \text{ satisfying a constraint } c \text{ on } V_1, \dots, V_m.$$

In the above symbolic state f_1, \dots, f_k are ground *cf* or *en* fluents and $val(o_1, V_1), \dots, val(o_m, V_m)$ are fluents that associate data objects o_1, \dots, o_m with their values V_1, \dots, V_m , respectively. Thus, a symbolic state represents the, possibly infinite, set of concrete

¹ The semantics presented in [9] supports the definition of unstructured workflows with arbitrary cycles, exceptional flows, and inclusive merge points, under the safeness assumption [2].

states that satisfy the given constraint. We say that a symbolic state S with associated constraint c is *subsumed* by another symbolic state T with associated constraint d , if S is equal to T , modulo variable renaming, and $c \sqsubseteq d$. We can often reduce the state space by avoiding to consider symbolic states that are *subsumed* by previously visited ones.

4.1 Enactment

We model the enactment of a DAPS as an *execution trace* (corresponding to a *plan* in the Fluent Calculus), i.e., a sequence of actions of the form $[act(a_1), \dots, act(a_n)]$ where act is either *begin* or *complete*.

The predicate $trace(S_1, T, S_2, N)$ defined below holds if T is a sequence of actions of maximum length N that leads from state S_1 to state S_2 :

$$\begin{aligned} trace(S_1, [], S_2, N) &\leftarrow N = 0 \wedge S_1 = S_2 \\ trace(S_1, [A|T], S_2, N) &\leftarrow N > 0 \wedge N_1 = N - 1 \wedge result(S_1, A, U) \wedge trace(U, T, S_2, N_1). \end{aligned}$$

In the following we use the abbreviation s_{0Proc} to denote the *initial* state of a process $Proc$, where the start event of $Proc$ is enabled to fire. Furthermore, we introduce the following rule to characterize the set of *final* states, where the end event of $Proc$ is enabled to fire

$$holds(final(Proc), S) \leftarrow bp(Proc, E_{start}, E_{end}) \wedge holds(en(E_{end}, Proc), S).$$

Our framework provides two services for analyzing process enactment, namely *trace compliance* and *simulation*.

(1) Trace compliance is the task of verifying whether an execution trace of a process is correct with respect to a given DAPS specification. Execution traces are commonly stored by BP management systems as process logs, representing the evolution of the process instances that have been enacted. Formally, a *correct trace* T of length N of a process $Proc$ is a trace that leads from the initial state to the final state of $Proc$, that is:

$$ctrace(T, Proc, N) \leftarrow trace(s_{0Proc}, T, S_f, N) \wedge holds(final(Proc), S_f)$$

The compliance of a trace with respect to a given DAPS can then be verified by evaluating a query of the form $\leftarrow ctrace(t, p, n)$ with respect to program \mathcal{KB} , where t is a ground list of length at most n and p is a process name. It is easy to see that such query terminates for every ground t , as the length of the second argument of the *trace* predicate decreases at each recursive call. An example of correct trace related to our running example is reported below.

```
[comp(s), comp(x1), comp(o1), beg(choose_prod), comp(choose_prod),
beg(choose_ship), comp(choose_ship), comp(o2), comp(x2), beg(add_item),
comp(add_item), comp(x3), beg(payment), comp(payment),
comp(x4), beg(shipment), comp(shipment), comp(e)]
```

The trace is guaranteed to be compliant also with respect to the data constraints associated with the DAPS, even if the information about the actual values of the data objects is not explicitly represented. A more complex representation of traces that also includes information about the data object values can easily be defined, but we omit it for reasons of simplicity.

(2) Simulation is the task of *generating* execution traces that represent possible process enactments. In order to analyze the dynamic behavior of the process in various situations, the process designer can analyze test cases where: (i) data objects are initialized to ground values in \mathbb{R} , and (ii) a subset of the database instance D is selected. A query of the form $\leftarrow \text{trace}(s_{0Proc}, T, S, n)$, where T is a free variable, can be used to generate the execution traces T of length not larger than n . The query terminates for every fixed integer n , as the last argument of *trace* decreases at each recursive call, and for each successful derivation from the query, the unification mechanism employed by CLP will bind T to a ground list of actions. Similarly, to the case of trace compliance, we can easily extend our definitions so as to generate traces that also contain explicit information about data values.

4.2 Symbolic Testing

Simulation is performed by selecting a finite set of test cases, that is, by fixing values for initializing the data objects and taking into consideration a specific database instance. However, the generation of test cases is not always straightforward. The mechanisms of symbolic computation provided by CLP (notably, unification and constraint solving) enable us to generate execution traces by only specifying constraints that those values are required to satisfy. Thus, we can initialize a data object to a value in a range, rather than to a concrete value. Furthermore, we can exploit the integrity constraints in the database schema DBS to perform a symbolic evaluation of a *trace* query without considering a fixed database instance.

For instance, in our eProcurement example, we can evaluate a *trace* query by initializing the data object *coupon* to a value X with $0 < X < 10$, as specified in Figure 1. Similarly, we can initialize *plafond* to a value X with $0 < X < 1000$. Furthermore, suppose that in our running example the integrity constraints relative to *product_price* and *ship_price* are:

$$\begin{aligned} \text{product_price}(X, P) &\rightarrow P > 5 \wedge P < 100 \\ \text{ship_price}(X, P) &\rightarrow P > 0 \wedge P < 15. \end{aligned}$$

Then we replace the database instance D in \mathcal{KB} by the inverse implications of these integrity constraints, that is, by the rules:

$$\begin{aligned} \text{product_price}(X, P) &\leftarrow P > 5 \wedge P < 100 \\ \text{ship_price}(X, P) &\leftarrow P > 0 \wedge P < 15. \end{aligned}$$

In general, for performing a symbolic testing task, we replace the ground facts that constitute the database instance D in \mathcal{KB} , by the inverse implications $p(X_1, \dots, X_n) \leftarrow c$ of all integrity constraints $p(X_1, \dots, X_n) \rightarrow c$ specified by DBS , hence deriving a new CLP program \mathcal{KB}' . Due to the least model semantics of CLP, we will have that $M(\mathcal{KB}') \models p(X_1, \dots, X_n) \leftrightarrow c_1 \wedge \dots \wedge c_k$, where c_1, \dots, c_k are the constraints implied by $p(X_1, \dots, X_n)$ in DBS . \mathcal{KB}' is an over-approximation of \mathcal{KB} , i.e., $M(\mathcal{KB}) \subseteq M(\mathcal{KB}')$. Then we evaluate a query of the form $\leftarrow \text{trace}(s_{0Proc}, T, S, n)$, for a given integer n . This query always terminates and, if it succeeds and returns an answer $T = t$, then there exists a database instance of DBS such that the DAPS $\langle WF, DC, DBS \rangle$ has t as a possible execution trace. The converse is not necessarily true, i.e., there may exist database instances that do not generate the trace t .

\mathcal{KB}' can be used to test reachability properties of the DAPS. For instance, the reachability of a *deadlock state* in n steps can be verified through a query of the form:

$$\leftarrow \text{trace}(s_{0Proc}, T, S_d, n) \wedge \neg r(S_d, S_n)$$

If the query succeeds, then the DAPS can reach, for some database instance, a potential deadlock state S_d and T is bound to a trace that leads to that state. If the query fails then, for any database instance satisfying the given database schema DBS , no deadlock is reachable in at most n steps.

In our example, two potential deadlock states are reachable by taking $n = 20$, both occurring when the control flow reaches *payment* (i.e., $\text{holds}(\text{en}(\text{payment}, eProc), S_d)$). In the first case the potential deadlock is due to a negative value of *amount* caused by a *coupon* value higher than the price of the product chosen by *choose_prod*, which prevents the execution of *payment*. Indeed, the following constraint associated with S_d is computed as an answer to the above query:

$$V_{\text{amount}} < 0 \wedge V_{\text{item_price}} < 0 \wedge V_{\text{coupon}} > 5$$

where V_o is the logical variable associated to the data object o in the state S_d through the fluent *val* (i.e., $\text{holds}(\text{val}(o, V_o), S_d)$). In the second case we have that the following constraint associated with S_d is computed as another answer to the above query:

$$V_{\text{amount}} = 0 \wedge V_{\text{item_price}} < V_{\text{plafond}} < 115$$

Here the potential deadlock is due to a value of *item_price* that exceeds the *plafond* granted to the user. For this reason, no item is added to the order, causing a zero value *amount* and preventing the execution of *payment*.

In order to prevent the above potential deadlocks it is sufficient to fix different ranges for the data objects *plafond* and *coupon* that make the two constraints unsatisfiable, e.g., $0 < \text{coupon} \leq 5$ and $115 \leq \text{plafond} < 1000$. In this way, the lowest *plafond* always covers the purchase of at least one product, and the highest coupon cannot exceed the price of any product.

4.3 Verification

Verification aims at checking whether a temporal property holds for all enactments of a given DAPS. Unlike the trace conformance, simulation, and symbolic testing tasks, for verification we do not assume a bounded trace length. Thus, due to the presence of data objects with values in \mathbb{R} and arbitrary, unstructured workflow graphs, the state space is infinite. Most verification problems, and in particular reachability, are undecidable in this setting. However, since we encode verification tasks as CLP queries, whose evaluation is based on a symbolic representation of states, by applying state subsumption we can avoid the actual exploration of the whole state space and terminate in many concrete verification examples.

(1) A very relevant behavioral property of a DAPS p is that, from any reachable state, it is possible to complete the process, i.e., reach the final state. This property, also known as *option to complete* [2], holds if the following query fails:

$$\leftarrow \text{reachable_state}(s_{0p}, S) \wedge \neg \text{reachable_final}(S)$$

where *reachable_final*(S) holds if a final state can be reached from S , i.e.,

$reachable_final(S) \leftarrow reachable_state(S, S_f) \wedge holds(final(p), S_f)$

In our running example, modified as suggested at the end of Section 4.2, the above query fails, hence enforcing the option to complete property.

(2) Another property that may reveal potential flaws in a DAPS is *executability* [11], according to which no activity reached by the control flow should be unable to execute due to some unsatisfied enabling condition. In our framework we can verify non-executability by the following query, which succeeds if it can be reached a state where some activity A is waiting for execution but its precondition is not enforced.

$\leftarrow reachable_state(s_{0p}, S) \wedge holds(cf(A_1, A, p), S) \wedge activity(A)$
 $\quad \wedge pre(C, A, p) \wedge \neg holds(C, S)$

In our running example we have a case of non-executability whenever *apply_coupon* and *choose_prod* are both scheduled for execution after $o1$, according to the inclusive gateways semantics. In this case, since *apply_coupon* requires a value of *item_price* greater than 0, it will not begin its execution until the completion of *apply_coupon*.

(3) Temporal queries can also be used for the verification of *compliance rules*, i.e., directives expressing internal policies and regulations aimed at specifying the way an enterprise operates. We report below two such compliance rules related to our running example. The first one requires that in any possible enactment the amount never reaches a negative value. The following query succeeds if it is possible to reach a state of the process where *amount* < 0.

$\leftarrow reachable_state(s_{0eProc}, S) \wedge holds(amount < 0, S)$

In our running example, modified to avoid deadlock as indicated at the end of Section 4.2, this query fails, thus enforcing the compliance rule.

A second example is a compliance rule requiring that it is possible to add an item to the order after the payment of part of the due amount. This property is encoded by the following query, which succeeds in our running example.

$\leftarrow reachable_state(s_{0eProc}, S_1) \wedge holds(en(payment, eProc), S_1)$
 $\quad \wedge reachable_state(S_1, S_2) \wedge holds(en(add_item, eProc), S_2)$

5 Implementation

We implemented the proposed framework in a proof-of-concept tool, extending the system discussed in [12, 9]. This extension provides an integrated environment to: *i*) edit BPs using the graphical BPMN editor shown in Figure 1, *ii*) annotate BP elements in terms of preconditions, effects, and guards *iii*) translate the annotated BPs into Constraint Logic Programming, and *iv*) handle the communication with an underlying inference engine that compiles the CLP program representing the DAPS and performs reasoning services through the querying mechanism.

The inference engine is built upon SWI Prolog² and is based on a suitable encoding of the set \mathcal{KB} of rules defined in Section 4. The constraints are handled by the built-in SWI solver for equality and inequality constraints over the reals. While the translation

² <http://www.swi-prolog.org/>

of \mathcal{KB} is straightforward, additional considerations are needed when dealing with the evaluation of queries that involve state reachability (i.e., the relation *reachable_state*). Indeed, similarly to many other Prolog engines, SWI generates derivations from a query by using a depth-first search strategy that never looks at the queries derived before the current one. For this reason query evaluation may enter an infinite loop in the presence of recursive rules. To mitigate this difficulty, at least partially, the definition of the predicate *reachable_state* (reported below) has been implemented through *memoing* [13], i.e., a strategy for storing intermediate results and avoiding to prove sub-goals more than once.

```
reachable_state(X,X).
reachable_state(X,Z) :-
    result(X,_,Y),
    constrained_state(Y,cs(Y1,CY)),
    \+ subsumed_by_visited(cs(Y1,CY)),
    assert(visited(cs(Y1,CY))),
    reachable_state(Y,Z).
```

where: *constrained_state*(*Y*, *cs*(*Y1*, *CY*)) holds if *Y1* is the set of fluents in *Y* and *CY* is the constraint associated with the variables in *Y*; *subsumed_by_visited*(*cs*(*Y1*, *CY*)) holds if a fact *visited*(*cs*(*W1*, *CW*)) belongs to the program, such that *Y1* is an instance of *W1* for some substitution and *CW* entails *CY*; *assert*(*visited*(*cs*(*Y1*, *CY*))) adds the fact *visited*(*cs*(*Y1*, *CY*)) to the program.

Essentially, during the evaluation of a goal including a *reachable_state* atom (i.e., requiring the state space exploration), for every reached state a subsumption test is performed, in order to verify whether a state that subsumes the current one has been already considered. If it is the case, the sub-goal fails, avoiding redundant computations. Otherwise, the state is added to the set of the visited states and the exploration proceeds.

By exploiting this simple memoing mechanism, we were able to verify many reachability properties of various versions of the eProcurement example, and in particular all properties presented in Section 4.3, which in principle require the exploration of an infinite state space.

6 Conclusions and Discussion

In this paper we presented a framework, grounded in Constraint Logic Programming, for reasoning on BP models represented through a *data-aware* extension of the popular BPMN notation. The behavioral semantics of a process is defined as a (possibly infinite) state transition system by following an approach derived from the Fluent Calculus. Data values have a symbolic representation based on arithmetic constraints, which allow us to specify data objects manipulations and database interactions in terms of preconditions and effects of the enactment of activities. We discussed the reasoning tasks the framework enables, dealing with enactment, testing and verification, and how they can be implemented through CLP engines.

Our work is related to a growing stream of research ([14–17]) dealing with an integrated view of the process and data perspective in BP modeling and verification

(see, e.g., [5] for a survey). In [14] decidability and complexity results are provided for the verification of artifact-systems, specified according to a variant of the artifact-centric model introduced by IBM [3, 4] and extended with data dependencies and arithmetic. The main result is the identification of classes of decidable and tractable artifact-systems, under particular restrictions on the interactions between control flow and artifacts. In [15] Data-Centric Dynamic Systems are introduced, through a formalism whose expressive power is equivalent to artifact-centric models, where arithmetic constraints among data objects are not explicitly addressed. Here, a process is described in terms of condition-action rules and the data layer is a relational database that is updated by actions execution. The verification of temporal properties given μ -calculus variants is addressed, and some decidability and computational complexity results are provided. With respect to the above works, our objective is more pragmatic, in that our formalization enables the implementation of a number of reasoning services by taking advantage of CLP engines specifically designed to deal with constraint solving. Since we do not pose any restrictions on the procedural description of processes, directly corresponding to BPMN diagrams, nor to data manipulation, in our setting verification is in general undecidable. However, by considering process runs of bounded length, many tasks with a practical relevance terminate, in particular when related to simulation and testing.

Several approaches have been recently proposed in the workflow community to take into account process data, extending consolidated results originally conceived for dealing with control-flow only. Among them, some related problems addressed in literature concern: run-time support for the enactment of data-aware process models [18]; verification of workflow models where the data-flow is also represented [19]; conformance checking of the execution logs of a BP with respect to its modeled behavior [20]. In contrast, we focus on a logic-based formalization of data-aware BPs to enable static analysis tasks in the presence of arithmetic constraints and data dependencies.

Finally, other approaches based on LP that are worth mentioning are [17, 16]. In [17] it is discussed a formalization of constraints dealing with the data-flow perspective, designed to extend declarative workflow specifications. The framework is grounded in the Event Calculus, and its LP implementation is intended to address a-posteriori analysis of process logs and runtime monitoring. In [16] it is presented an approach to BP verification based on an extension of answer set programming with temporal logic and constraints, where the compliance of business rules is checked by bounded model checking techniques extended with constraint solving for dealing with conditions on numeric data. With respect to our setting, the framework assumes finite domains for variables besides several restrictions on the workflow, resulting in a less expressive (decidable) language, whose verification can be reduced to finite-state analysis.

The preliminary results presented in this paper open up several directions for future research. First of all, we plan to push forward the empirical evaluation of our proposal in each application scenario reported in Section 4. To this end, a relevant aspect to be further elaborated regards the adoption of program optimization techniques to enhance the performances of the reasoning approach. On a more theoretical perspective, we are investigating the class of BPs for which a symbolic finite-state space can be computed, in order to characterize the decidability and complexity of temporal verification tasks.

References

1. OMG: Business Process Model and Notation. <http://www.omg.org/spec/BPMN/2.0> (2011)
2. van der Aalst, W.M.P.: The application of Petri nets to workflow management. *Journal of Circuits, Systems, and Computers* **8**(1) (1998) 21–66
3. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. *IBM Syst. J.* **42**(3) (July 2003) 428–445
4. Hull, R.: Artifact-centric business process models: Brief survey of research results and challenges. In: *On the Move to Meaningful Internet Systems: OTM 2008, Part II*. Volume 5332 of LNCS. Springer (2008) 1152–1163
5. Calvanese, D., De Giacomo, G., Montali, M.: Foundations of data-aware process analysis: A database theory perspective. In: *Proceedings of the 32nd Symposium on Principles of Database Systems. PODS '13*, ACM (2013) 1–12
6. van der Aalst, W., Weske, M.: Case handling: A new paradigm for business process support. *Data Knowl. Eng.* **53**(2) (2005) 129–162
7. Thielscher, M.: Introduction to the Fluent Calculus. *Electron. Trans. Artif. Intell.* **2** (1998) 179–192
8. Jaffar, J., Maher, M.: Constraint logic programming: A survey. *Journal of Logic Programming* **19/20** (1994) 503–581
9. Smith, F., Proietti, M.: Rule-based behavioral reasoning on semantic business processes. In: *Proceedings of the 5th Int. Conf. on Agents and Artificial Intelligence, Volume II*, SciTePress (2013) 130–143
10. Lloyd, J.W.: *Foundations of logic programming*. Springer-Verlag New York, Inc. (1987)
11. Weber, I., Hoffmann, J., Mendling, J.: Beyond soundness: On the verification of semantic business process models. *Distrib. Parallel Databases* **27** (2010) 271–343
12. Smith, F., Missikoff, M., Proietti, M.: Ontology-Based Querying of Composite Services. In: *Business System Management and Engineering*. Volume 7350 of LNCS. Springer (2010) 159–180
13. Dietrich, S., Fan, C.: On the completeness of naive memoing in Prolog. *New Generation Computing* **15**(2) (1997) 141–162
14. Damaggio, E., Deutsch, A., Vianu, V.: Artifact systems with data dependencies and arithmetic. *ACM Trans. Database Syst.* **37**(3) (September 2012) 22:1–22:36
15. Bagheri Hariri, B., Calvanese, D., De Giacomo, G., Deutsch, A., Montali, M.: Verification of relational data-centric dynamic systems with external services. In: *Proceedings of the 32nd Symposium on Principles of Database Systems. PODS '13*, ACM (2013) 163–174
16. Giordano, L., Martelli, A., Spiotta, M., Dupré, D.T.: Business process verification with constraint temporal answer set programming. *TPLP* **13**(4-5) (2013) 641–655
17. Montali, M., Chesani, F., Mello, P., Maggi, F.M.: Towards data-aware constraints in Declare. In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing. SAC '13*, ACM (2013) 1391–1396
18. Meyer, A., Pufahl, L., Fahland, D., Weske, M.: Modeling and enacting complex data dependencies in business processes. In: *Proceedings of 11th International Conference on Business Process Management, BPM 2013*. Volume 8094 of LNCS. Springer (2013) 171–186
19. Sidorova, N., Stahl, C., Trcka, N.: Soundness verification for conceptual workflow nets with data: Early detection of errors with the most precision possible. *Inf. Syst.* **36**(7) (2011) 1026–1043
20. de Leoni, M., van der Aalst, W.M.P.: Aligning event logs and process models for multi-perspective conformance checking: An approach based on integer linear programming. In: *Proceedings of 11th International Conference on Business Process Management, BPM 2013*. Volume 8094 of LNCS. Springer (2013) 113–129

CoPrA2GO: An App for Coding Collaboration Processes

Emmanuel Nowakowski

University of Innsbruck, Innsbruck, Austria
emmanuel.nowakowski@student.uibk.ac.at

Isabella Seeber

University of Innsbruck, Innsbruck, Austria
isabella.seeber@uibk.ac.at

Ronald Maier

University of Innsbruck, Innsbruck, Austria
ronald.maier@uibk.ac.at

Fulvio Frati

Università degli Studi di Milano, Milan, Italy
fulvio.frati@unimi.it

Abstract. Organizations seek ways how to support teams in their communication but face challenges how to make communication processes measurable and visible. Past communication research came up with numerous interaction coding systems enabling the analysis of communication. However, today we see only few applications in research and practice due to the labor-intensive effort connected to coding and analysis. This paper addresses this problem and introduces the iPad application CoPrA2GO that strives to make the coding and analysis of communication more convenient and applicable for researchers and practitioners. A user acceptance test was conducted involving 4 IS graduate students coding 23 team meetings in real-time. Our findings suggest that CoPrA2GO is useful for coding communication in real-time and providing feedback immediately after meetings.

Keywords: communication analysis, CoPrA, real-time, CoPrA2GO, team performance

1 Introduction

Worldwide organizational structures have been transformed from work organized around individual jobs to team-based work structures [1]. The transformation is mainly attributed to pressures created by increasing global consolidation, innovation, and competition. There is a need for more rapid, adaptive, and flexible responses to overcome

these pressures [2]. Additionally, problems are more and more complex, so that no single individual has enough influence, resources, or expertise to solve the problem alone. Therefore, team collaboration has become an omnipresent feature of organizational life [3]. Given this, it is important for organizations that teams work effectively. According to Pentland [4] it is possible to predict a team's success by looking at communication data generated during collaboration. This stream of research recommends automated ways of gathering and analyzing communication by e.g., calculating the frequency of interaction exchanges [4]. Yet, it is not adequate to capture the meaning of communication. Several interaction analysis approaches [e.g., 5, 7, 8, and 9] have been suggested to gather and analyze communication and its content. All of these approaches require the manual creation of communication logs to deduce the meaning of communication [7, 8, and 9]. Interaction coding systems such as IPA [8], SYMLOG [16], DFCS [9], and TEMPO [15] are usually independent of software tools and consequently additional information in relation to the communication act such as timestamp or name of the team member needs to be included by hand. For further analysis additional extra human manipulation is required to transfer communication logs into tools for analysis to calculate, e.g., depth or breadth of discussions and participation. Consequently, researchers and practitioners are faced with labor-intensive and time-consuming efforts to analyze team communication [5]. Better methods are needed that allow analyzing team behavior and team performance in a discreet, flexible and real-time manner [10, 11].

This paper strives to contribute to this call for better methods and presents a design artifact, which enables IT-supported coding of communication. This design artifact is implemented as an iPad application, named CoPrA2GO, which adopts a range of communication acts as defined in the COllaboration PRocess Analysis (CoPrA) technique [5]. The app allows for real-time coding of communication and is interfaced with the CoPrA Tool [12] for process analysis. We evaluated the usefulness of the tool by conducting a laboratory experiment in which four IS graduate students coded communication of 23 teams working on a decision-making task in real-time. The coders were interviewed in a subsequent focus group interview to gain insights into their experience with using the tool. The following three research questions were stated to assess the acceptability of the tool: (1) *For which purpose is CoPrA2GO used?*, (2) *What makes CoPrA2GO useable?*, and (3) *In which settings could CoPrA2GO be used?*.

The paper is structured as follows. Section 2 provides the background on team effectiveness and communication coding. Section 3 introduces the CoPrA2GO functionality. Section 4 describes the laboratory experiment and the focus group interview in which the application was tested for user acceptance. Section 5 reports the results of the CoPrA2GO user acceptance test. Finally, Section 6 and 7 contain our discussion and conclusion.

2 Background

Teams gain more and more importance and so does measuring team effectiveness [2, 3]. According to Cohen and Bailey [13] effectiveness can be measured along three di-

mensions comprising performance outcomes (e.g. productivity, response times, innovation), behavioral outcomes (e.g. turnover, absence, safety), and member attitudes (e.g. commitment, employee satisfaction, trust in management). Team performance is context specific, which makes it difficult to define criteria that are generalizable to other teams, the organization or even beyond [6]. In this paper, team effectiveness is primarily based on team performance composites, which are shaped by team behavior patterns evolving through social interaction. Firstly, because it is considered as important in organizational behavior and human resource management literature [14]. Secondly, because it is observable with the help of automated tools, and thirdly, because we need better measurement of factors influencing team effectiveness [11].

For the automated analysis of team performance based on communication, Pentland [4] proposed a composite of three performance criteria that he refers to as energy, engagement and exploration. Where energy denotes the individual participation in the team, engagement the communication of team members within the team, and exploration the communication of team members with other teams. Additionally, Pentland [4] states that the most important factor for high performing teams is the balance between energy and engagement.

There are long-established approaches for the analysis of team behavior based on communication. Examples are the Interaction Protocol Analysis (IPA) [8], the Decision Functioning Coding System (DFCS) [9], the TEMPO system [15], and the SYMLOG methodology [16]. For these traditional approaches, the researcher first needs to transcribe video and/or audiotaped communication, perform a coding procedure on the transcripts, and analyze the communication logs that resulted from the previous step. Each code describes the meaning of the underlying information [17]. The whole procedure aims at reducing the complexity of the team's communication to a simpler set of categories [18]. Each of the above mentioned approaches differ to a certain extent to the kind of team behavior they can deduce. For example, the IPA framework [8] includes categories for coordination and emotions and is, like the DFCS [9], capable of coding task-related communication. SYMLOG [16] and TEMPO [15] are further developments of IPA [8] and allow to create team member profiles and distributions of team behavior (for taskwork and teamwork), respectively. A further advancement in communication analysis is represented by the CoPrA technique. The technique differs from other interaction analysis methods in such sense that it puts emphasis on aggregating communication to topics during its data preparation phase [19].

3 Artifact Description: CoPrA2Go App

The basis of our work builds upon the aforementioned CoPrA technique. The CoPrA2GO application supports two major activities of collaboration analysis, comprising real-time coding of communication (data preparation) and mining patterns of team behavior (data analysis). Its main functionality is to facilitate real-time coding of communication. CoPrA2GO adopts the coding schema of the CoPrA technique to ensure compatibility with the CoPrA Tool. CoPrA2GO consists of five screens, in Objective-C so-called view controllers, which guide the coder from the login screen to the

coding screen. During coding, the app generates an MXML file in the background, which allows to be analyzed by the CoPrA Tool for mining team behavior patterns and whose results are returned to the iPad screen (see Fig. 1). Please refer to [12] for more information on the mining of team behavior patterns. The remainder of the section describes the intended use of CoPrA2GO and its interface to the CoPrA Tool.

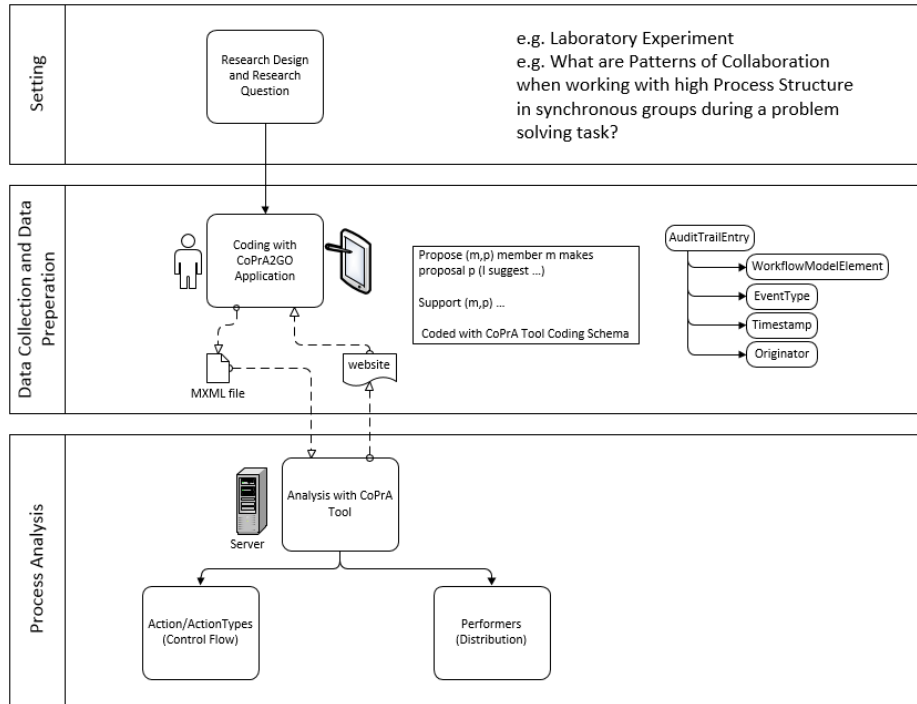


Fig. 1. CoPrA2GO for Real-Time Analysis.

Pre-Conditions

A pre-condition of using CoPrA2GO is that any coder is familiar with the communication actions that can be used for coding communication. The code book contains 18 codes including codes such as propose idea, ask for clarification, or support idea (see Appendix for a more detailed description). In addition, all coders need to be familiar with the app interface so that the respective codes can be quickly found. This ensures the on-the-fly interpretation of communication is possible and does not hinder the use of CoPrA2GO by longer thinking about code meanings and location.

Observer Registration View Controller

The design goal is that the user engagement with the application is aided by a visual representation of a meeting that looks like a physical meeting room. Therefore, the first view controller looks like a doorplate and is the entrance to the room. The coder has to register by entering their name (or a nickname) in the text field, to make it traceable

who coded the specific collaboration session. After the registration is finished, the coder is able to continue to the next view controller by hitting the submit button.

Choose Table Design View Controller

In this view the coder has to choose one of four table designs. Together with the next step, group setup view controller, the coder recreates a virtual environment that reflects the physical setting including team members, facilitator, and meeting table. This is considered to ease the cognitive load for coding communication as the drag-and-drop of communication acts onto team members or the facilitator on the iPad screen is similar to their location in the physical environment.

Group Setup View Controller

After that, the coder is forwarded to the group setup view controller, used for the MXML file generation. Here, the coder enters the names of the facilitator and the team members for later placement in the virtual meeting room. Furthermore, the coder includes the server address of the CoPrA Tool server for sending the MXML for further analysis. The number of team members is currently limited to six plus one facilitator. The number of team members could be easily adapted by adjusting the underlying array, table view, and restriction. The data collected here is saved and passed to the observation view controller.

Observation Screen View Controller

Fig. 2 depicts the observation screen in which the communication coding takes place. We first describe the elements seen on the screen identified by 1 to 7 in Fig. 2 and then the actions that can be coded. In particular, (1) shows the text field where the Task ID has to be entered, (2) depicts the member labels, initially placed on the left and right hand side of the screen, which have the names specified in the group setup view controller, (3) shows the play/pause mechanism for the timer functionality, (4) depicts all the 18 code buttons from the CoPrA code book, (5) shows the activity stream that contains the last seven coded actions, (6) points out the *undo* and *redo* mechanism for the activity stream, which has also effects on the resulting MXML file, and finally, (7) shows the facilitator, who is guiding the team members through the collaboration session.

To be able to start the coding procedure, the coder first has to enter the Task ID to make communication acts assignable to a specific task. The second step is to drag-and-drop the representations of the team members on their actual sitting position at the table. After these steps are fulfilled, the play button for the timer, at the bottom on the left side, has to be hit to enable the assignment of timestamps to any coded communication action. Once the timer runs, the actual coding activity can begin.

Each code button (4) can be moved per drag-and-drop and is able to detect collisions with the member labels. Therefore, if for example the provision button collides with the label of *member1* (2), it is logged that *member1* performed, e.g., a provision action at a specific time. Furthermore, the facilitator (7), displayed as the figure behind the desk, has also a collision detection. After each code assignment, the activity stream (5)

adds the latest code assignment at the top of the list and moves the older one field below. Additionally, with the undo and redo buttons (6) on the right side, it is possible to delete the last performed actions, if an error occurred, or to restore the last deleted action. Furthermore, the Task ID (1) can be changed during the coding procedure to ensure that after a new task has started, the corresponding collaboration codes are assigned to the correct task, e.g., evaluation or idea. That has the effect of generating a new “ProcessInstance” in the MXML file. The definition of task and the corresponding Task ID has to happen before the actual use of the application.

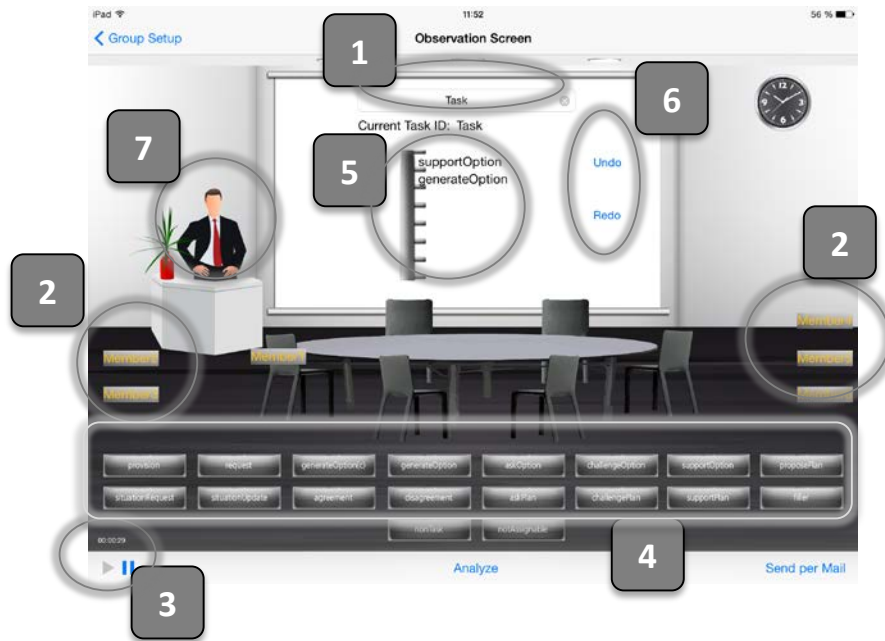


Fig. 2. Observation Screen View Controller.

After the collaboration session is finished, the pause button (3) has to be hit, in order to activate the “Analyze” and “Send per Mail” buttons. This is implemented to prevent errors, which could happen by accidentally tapping the “Analyze” button during the coding procedure.

At the end of the coding, the coder can choose between sending the generated MXML file per email to further analyze it by using, e.g., ProM [20], or to send it directly to the CoPrA Tool server for further analysis.

Web View Controller

This controller will be activated when the analyze option in the observation screen view controller is chosen. At this point, the connection to the CoPrA Tool is established and the MXML file is sent to the server for further analyses. In this controller, the coder can choose which analysis methods to perform on the MXML files sent to the server.

In fact, the CoPrA Tool applies a set of predefined metrics to the communication logs and allows the visualization of results, both in distribution and flow perspectives. The results are then shown in the CoPrA2GO app in form of tables or graphs.

4 Method

This paper describes a design science research study reporting on the activities of the research cycles comprising rigor cycle, design cycle, and relevance cycle [21]. In May 2014, the CoPrA2GO tool was tested in the context of a laboratory experiment at the University of Innsbruck collecting data from 92 undergraduate students that were randomly assigned to 23 teams. The collaboration task is named Norvos and represents a decision making task. The task was adapted from an existing one and developed further to better fit a student context. It is about a flooding that hit the city of Norvos. The goal of collaboration is to decide on supporting measures to deal with the aftermath of the flooding (e.g. organizing additional support of water and food, providing medical personnel and supplies, assisting in the repair of infrastructure, and supplying general clothing and shelter). Each session was guided by one of five facilitators. In addition, one (of four) IS graduate students joined the session as testers to evaluate CoPrA2GO by observing team discussions and performing real-time communication coding. They received a short training on how to use CoPrA2GO and were handed-out an adapted version of the CoPrA code book explaining each code of a communication act. Additionally, testers were asked to write down instant feedback on "what works well", "what does not work well", and "what catches your eye in general". After they finished their coding, they sent the resulting MXML file per email, for backup reasons, and to the CoPrA-Tool test-server, for analyzing it directly.

Additionally, the testers were asked to jointly reflect on their experiences in a focus group interview about their perceptions on the usefulness of CoPrA2GO. The interview lasted for 30 minutes, and it was videotaped and transcribed afterwards. The transcription was analyzed by applying the coding procedure and method of Corbin et al. [22] using ATLAS.ti [23]. This procedure consists of open, axial, and selective coding. The goal of open coding is to break down the data analytically by an interpretive process. This should help to gain new insights on the data and generate subcategories. The goal of axial coding is to relate coding categories to their subcategories and to test these relationships against the data. During this coding phase the categories are also developed further. Finally, the goal of selective coding is to unify all categories, found in the phase before, around a central "core" category, where the core category represents the central phenomenon of the study [22]. Additionally, the sheets for instant feedback were matched with the interview answers to enrich them.

5 Results of the User Acceptance Test

The aim of the qualitative content analysis was to test the user acceptance of the CoPrA2GO application on the basis of three research questions comprising (1) *For which purpose is CoPrA2GO used?*, (2) *What makes CoPrA2GO useable?*, and (3) *In*

which settings could CoPrA2GO be used? We tested the user acceptance based on the criteria of the technology acceptance model [26], where primarily perceived ease-of-use and perceived usefulness lead to user acceptance of technology.

When addressing research question (1), the results show that three out of four testers see the purpose of CoPrA2GO in providing feedback immediately after the meeting. All of the testers agreed on its usefulness for recognizing team behavior patterns and performing descriptive statistics. In this context, they mentioned that it is very interesting to see how many ideas were generated (descriptive statistics), and which ideas got immediately challenged or supported (team behavior patterns).

“I think that at the end you can easily see who has contributed the most. Especially, the number of generated options of each team member. You can also see the participation of each member.” – Tester 1

“Additionally, you are able to see which option is immediately challenged or supported. Therefore, it is nice for pattern recognition.” – Tester 2

Testers also mentioned some disadvantages for using CoPrA2GO. The application cannot compensate the reflection cycles that are common to traditional qualitative content coding. That is mainly because the traditional coding of audio or videotaped communication happens at a later point in time, is not as time-pressured, and options for discussing specific codes exists. This is believed to lead to a higher precision of coding.

“It is really fast, but it cannot be compared with manual communication coding, because the level of detail of manual coding cannot be reached. During manual coding you have time to think about the communication log, to discuss about the codes you would like to apply, you are able to perform an intercoder reliability, and more. – [CoPrA2GO] is another approach, a simpler and faster one.” – Tester 2

When addressing research question (2), the results show what aspects make CoPrA2GO usable for real-time coding of communication in synchronous, small sized, face-to-face settings. All testers agreed on the clear, understandable, simple, and intuitive interaction with the application and hence described it as an easy to use application.

“It was very easy to interact with the application, the drag-and-drop functionality worked perfectly and the coding never failed.” – Tester 4

“[...] The application was simple to use, it was obvious what was to do, and yes, it was very intuitive.” – Tester 1

Additionally, the testers agreed on a quick learning phase while using CoPrA2GO. For instance, just one out of four testers had problems with using the application in the first coding session. This could also be seen in the log file where the first session consisted of 20 collaboration act entries with six codes used and the second consisted already of 167 entries with 12 codes used. Thus, he/she needed just 30 minutes to learn how the application should be used.

“In the first session I was not used to the application and the setting, so I was just able to code on a coarse grained level of abstraction. [...] In the following sessions I was able to code in much more detail, because I knew the setting of the experiment and I was used to the application.” – Tester 1

They also mentioned that it is an interesting coding approach as the efforts inherent to traditional coding are low. With the above outlined constraints, it represents an easy way to code communication.

“An advantage is that you get results without the need of manually coding communication.” – Tester 4

When addressing research question (3), testers gave opinions about the setting in which they deem CoPrA2GO as useful. The testers stated that the team’s communication needs to be well-structured. The reason for this is that there exist problems with the real-time coding of parallel communication, when just one coder has to code the communication of a small group. Another reason is that the communication should not be too fast-paced, otherwise the coder is too slow to code every aspect of the collaboration session.

“I think that the application is very useful in [...] moderated setting, for example, Tester 1, then Tester 3, and finally, I say a sentence.” – Tester 2

“To code a discussion after a presentation would be possible.” – Tester 4

“Yes, and a podium discussion would be a good example.” – Tester 2

“[...] But I could imagine, that in a business meeting, where real discussions happen, it could be very hard to assign the codes [with CoPrA2GO].” – Tester 2

“Yes, because there is parallel communication and discussions are too fast paced to code for one coder. I would propose to use at least two coders for four team members, or for every team member one real-time coder.” – Tester 4

Table 1 summarizes the results of the user acceptance test. The first column includes the research question and the second column provides the answers to each research question that were derived from qualitative content coding.

Table 1. Results of the User Acceptance Test

WHAT purposes is CoPrA2GO used for?	<ul style="list-style-type: none"> • recognizing team behavior patterns • providing quantitative team output statistics
WHAT makes CoPrA2GO usable?	<ul style="list-style-type: none"> • simplicity • easy-to-learn • no extra manual coding • providing feedback immediately after the meeting
WHERE CoPrA2GO could be used?	<ul style="list-style-type: none"> • well-structured communication • little parallel communication • limited speed of communication

6 Discussion and Limitations

In this paper we presented CoPrA2GO, an iPad application, for real-time communication coding, which offers the possibility to get feedback on collaboration processes immediately after the session. In the previous section we presented the results of our user acceptance test (summarized in Table 1), which will now be discussed, again structured along the three research questions we posed in the introduction of this paper.

1. For what purposes is CoPrA2GO used?

The user acceptance test showed that the application is a tool that is mainly useful for receiving feedback on team behavior patterns and outputs, and giving the possibility to analyze the generated log files immediately after the meeting. This is possible because each communication act is combined with a timestamp and is stored in chronological order in the communication log, and, in particular, because CoPrA2GO is connected to the CoPrA Tool [12] in the back-end. Additionally, the MXML log files have a structure that can also be analyzed with ProM, which provides additional process metrics that could be applied [20]. Many teams do not have the necessary communication ability to guide their team members or keep their interactions effective [24]. Feedback can mitigate this problem because it allows to draw the team's attention to the task or the group, hence, affecting behavior [25]. Additionally, it is possible to code while being fully aware of the context of the communication and the non-verbal communication in the room, which makes it possible to gain deeper insights in the communication than looking at a simple communication log ex-post. Additionally, CoPrA2GO could also be used to analyze video and/or audiotaped collaboration sessions ex-post. Furthermore, this ex-post approach could be used to refine an initial real-time coding.

2. What makes CoPrA2GO usable?

CoPrA2GO is usable because of its simplicity, clearness, and intuitiveness. The application is easy-to-use and easy-to-learn, because its design is perceived as user-friendly. Specifically, it does not allow erroneous user inputs and offers just the necessary input possibilities. Furthermore, the drag-and-drop functionality is already well established in software, like operating systems and mobile applications, which makes the effort of getting used to it low. The quick learning process is on the one hand influenced by the just mentioned user friendliness and on the other hand by the obviousness of the application usage. Alongside this, coding happens quickly and is relatively effortless, compared to common qualitative coding practices. This is mainly due to the reason that the coding happens on-the-fly during the meeting and is IT-supported, i.e. CoPrA2GO. Additionally, the analysis happens at the back-end with help of the CoPrA Tool which enables CoPrA2GO, unlike other coding systems, to run analysis immediately after coding without extra effort related to the insertion of data in spreadsheets, conversion of log files, or switching consciously between systems.

3. In which settings is CoPrA2GO useful?

The results show that CoPrA2GO is especially a tool for teams having well-structured communication. There were two aspects mentioned, which could hinder the use of the application, namely, parallel and too fast-paced communication. For this reason there is a need of either a facilitator who moderates the collaboration session, or a self-managed team that selects a leader to coordinate their processes. The facilitator's job is to manage the meeting effectively, to handle group dynamics, and to use adequate technology [27]. A facilitator may intervene into the content, process, or how technology should be used [28]. To overcome the issues of parallel and too fast-paced communication, especially, the process facilitation part is interesting. Process facilitation helps a team to manage and coordinate collaboration activities [29]. As a result, the structure of the overall process is improved, for example, by agreeing on interaction routines [28]. Furthermore, this well-structured communication is needed because, according to Cognitive Load Theory [30], the working memory of humans is limited in capacity when it has to process new information. Therefore, it could happen that parallel and too fast-paced communication lead to an overload of cognitive capacity, which can result in a decrease of the overall performance [31]. In our conducted experiment the testers had no issues with cognitive load and, therefore, were capable of keeping up with what was said and of using CoPrA2GO for real-time coding, which was indicated by the user acceptance test. According to the model of technology acceptance [26] it is likely that CoPrA2GO will find acceptance as it is perceived as useful and easy-to-use.

However, there also exist limitations that should be considered. Firstly, the application used during the real-time coding testing scenario is just a stable prototype. Secondly, the number of testers was limited to four, which could lead to a bias in the user acceptance test. Finally, the user acceptance test is based on a focus group interview, where maybe additional single interviews reveal different opinions.

7 Conclusion

This paper introduced CoPrA2GO, an application suitable for real-time coding of communication of small teams in face-to-face settings. The user acceptance test shed light on the perceived purpose of the tool, in which settings it could be used and why its use is perceived as effortless. There exist topics for future research that should be considered. Firstly, the limited code book does not allow tracking socio-emotional cues, such as mood or specific body language. Especially during real-time coding of a collaboration session, the coder has the possibility to see such behavior and assign it to a team member. Also tracking the mood of each team member and the general mood within the team would be an interesting addition for the analysis of team performance. This could be done by adding a mood barometer to the application that reacts interactively when a change in mood is coded. In fact, recent studies [32] demonstrate that it is indeed very difficult to understand individual and team emotions. Even if it is possible to detect emotion of an individual by analyzing video registration it is very difficult, because these emotions are closely related to the actual context that will influence the interpretation of facial and body signals and movements. Therefore, a coder who is present in

the collaboration session and aware of the context, is able to detect the general mood and will help to understand also this aspect of collaboration. For this purpose, the MXML schema would also need a revision, adding a specific tag and properties to save mood information. Furthermore, the CoPrA Tool would need new metrics for mood evaluation, and to combine mood with the overall team performance. Secondly, the user acceptance test was performed in small team settings. It would be interesting to validate the usefulness of CoPrA2GO in bigger teams. One challenge could be that a single CoPrA2GO coder might not be able to handle the cognitive load of real-time communication coding of bigger teams.

As result the paper has implications for research and practice. On the one hand, the paper contributes to research because CoPrA2GO represents a tool for IT-supported communication coding with less demand on time and labor than traditional coding system. This should benefit further advances in our research on team effectiveness and collaboration analysis on the basis of communication. On the other hand, it contributes to practice by providing an easy-to-learn and easy-to-use tool for real-time coding and collaboration process feedback immediately after meetings. Consequently, it gives the possibility to get better measurements on teamwork and team performance [11].

Appendix

Table 2. CoPrA2GO Code Book; short forms are in brackets.

<i>Code</i>	<i>Description</i>
Information / Knowledge Provision (provision)	Occurs when someone provides clarifications for problem analysis
Information / Knowledge request (request)	Occurs when someone asks for clarifications of the problem analysis, or for repetition of immediately preceding information
Option Generation – Complete (generateOption(c))	Statements explicitly proposing a complete or near complete solution on the basis of parts of solutions and statements that close the discussion on a specific idea
Option Generation – Partly (generateOption)	Statements that provide an incomplete solution and are part of the generation and evolution of an idea
Ask Option (askOption)	Occurs when someone asks for a response to a proposed option which is part of the problem-solving task
Option Challenge (challengeOption)	Occurs when someone provides criticism of a single potential proposed idea (option) in the problem-solving task environment
Option Support (supportOption)	Occurs when someone provides support to a proposed idea, a partly proposed option or to a complete option by providing an argument for the option

Plan Propose (proposePlan)	Utterances that suggest (1) to move on in the team process or (2) to alter the team process by including a further team process step
Plan Support (supportPlan)	Utterances that give reasoning why to support a proposition made for a process or plan regulation
Plan Challenge (challengePlan)	Utterances that give reasoning why to challenge a proposition made for a process or plan regulation
Plan Ask (askPlan)	Question utterances asking where, when, why, who, and how should proceed with the team process
Situation Update (situationUpdate)	Statements that provide information about what the team is currently doing, or what it is currently happening, both on process and task level
Situation Request (situationRequest)	Statements that ask about what the team is currently doing or what is currently happening with the task
Agreement / Disagreement	Expressions of agreement or disagreement with no rationale provided.
Incomplete / Filler (filler)	Utterances that cannot be categorized into one of the other categories because statements are incomplete or just fillers
Non Task (nonTask)	Utterances that signal joking or that are out of the topic of the task
Not Assignable (notAssignable)	In this case no communication action can be associated to a thought unit

References

1. Lawler, E.E., Mohrman, S.A., Ledford, G.E.: Creating High Performance Organizations: Practices and Results of Employee Involvement and Total Quality Management in Fortune 1000 Companies. Jossey-Bass, San Francisco (1995)
2. Kozlowski, S.W.J., Bell, B.S.: Work Groups and Teams in Organizations. In: Borman, W.C., Ilgen, D.R., Klimoski, R.J. (eds.) Handbook of Psychology. Industrial and Organizational Psychology, vol. 12, pp. 333-375. Wiley-Blackwell, New York (2003)
3. De Vreede, G., Briggs, R.: Collaboration Engineering: Foundations and Opportunities. Journal of the Association for Information Systems: Editorial to the Special Issue on the Journal of the Association of Information Systems. 10(3), 121-137 (2009)
4. Pentland, A.: The New Science of Building Great Teams. Harvard Business Review. April, 60-70 (2012)
5. Seeber, I., Maier, R., Weber, B.: Macrocognition in Collaboration: Analyzing Processes of Team Knowledge Building with CoPrA. Group Decis Negot. 22(5), 915-942 (2013)

6. Mathieu, J., Maynard, M. T., Rapp, T., Gilson, L.: Team Effectiveness 1997-2007: A Review of Recent Advancements and a Glimpse into the Future. *Journal of Management*. 34(3), 410-476 (2008)
7. Keyton, J., Beck, S.J.: The Influential Role of Relational Messages in Group Interaction. *Group Dynamics: Theory, Research, and Practice*. 13(1), 14-30 (2009)
8. Bales, R.F.: *Interaction Process Analysis: A Method for the Study of Small Groups*. Addison-Wesley, Oxford (1950)
9. Poole, M.S., Holmes, M.E.: Decision Development in Computer-Assisted Group Decision-Making. *Human Communication Research*. 22(1), 90-127 (1995)
10. McCowan, I., Gatica-Perez, D., Bengio, S., Lathoud, G., Barnard, M., Zhang, D.: Automatic Analysis of Multimodal Group Actions in Meetings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 27(3), 305-317 (2005)
11. Salas, E., Cooke, N. J., Rosen, M.a.: On Teams, Teamwork, and Team Performance: Discoveries and Developments. *Human Factors: The Journal of the Human Factors and Ergonomics Society*. 50(3), 540-547 (2008)
12. Frati, F., Seeber, I.: CoPrA: A tool for Coding and Measuring Communication in Teams. 2013 7th IEEE International Conference on Digital Ecosystems and Technologies (DEST). 43-48 (2013)
13. Cohen, S.G., Bailey, D.E.: What Makes Teams Work: Group Effectiveness Research from the Shop Floor to the Executive Suite. *Journal of Management*. 23(3), 239-290 (1997)
14. Bommer, W.H., Johnson, J.L., Rich, G.A., Podsakoff, P.M., MacKenzie, S.B.: On the Interchangeability of Objective and Subjective Measures of Employee Performance. *Personnel Psychology*. 48(3), 587-605 (1995)
15. Futoran, G.C., Kelly, J.R., McGrath, J.E.: TEMPO: A Time-Based System for Analysis of Group Interaction Process. *Basic and Applied Social Psychology*. 10(3), 211-232 (1989)
16. Keyton, J., Wall, V.D.: Symlog: Theory and Method for Measuring Group and Organizational Communication. *Management Communication Quarterly*. 2(4), 544-567 (1989)
17. Myers, M.D.: *Qualitative Research in Business & Management*. SAGE Publications Limited (2008)
18. Patton, M.Q.: *Qualitative Evaluation and Research Methods*. Sage Publication. Thousand Oaks, California (2002)
19. Seeber, I., Maier, R., Ceravolo, P., Frati, F.: Tracing the Development of Ideas in Distributed, IT-Supported Teams during Synchronous Collaboration. *Proceedings of the 22nd European Conference on Information Systems*. Tel Aviv, Israel. 1-17 (2014)
20. van der Aalst, W., van Dongen, B., Günther, C., Mans, R., de Medeiros, A., Rozinat, A., Rubin, V., Song, M., Verbeek, H., Weijters, A.: *ProM 4.0: Comprehensive Support for Real Process Analysis*. Springer Verlag, Berlin (2007)
21. Hevner, A.R.: A Three Cycle View of Design Science Research. *Scandinavian Journal of Information Systems*. 19(2), 87-92 (2007)
22. Corbin, J., Strauss, A., Clarke, A., Gerhardt, U., Glaser, B.: Grounded Theory Research: Procedures, Canons and Evaluative Criteria. *Qualitative Sociology*. 13(1), 3-21 (1990)
23. Muhr, T., Friese, S.: *User's Manual for ATLAS.ti 5.0*. ATLAS. ti Scientific Software Development GmbH. Berlin (2004).
24. Keyton, J., Beck, S. J., Beth, M.: Macrocognition: A Communication Perspective. *Theoretical Issues in Ergonomics Science*. 11(4), 272-286 (2010)
25. Kluger, A., & DeNisi, A.: The Effects of Feedback Interventions on Performance: A Historical Review, a Meta-Analysis, and a Preliminary Feedback Intervention Theory. *Psychological Bulletin*. 112(2), 254-284 (1996)

26. Davis, F. D.: Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly*. 13(3), 319–340 (1989)
27. De Vreede, G.-J., Boonstra, J., Niederman, F.: What Is Effective GSS Facilitation? A Qualitative Inquiry into Participants' Perceptions. *Proceedings of the 35th Annual Hawaii International Conference on System Sciences, HICSS*. 616-627 (2002)
28. Seeber, I., Maier, R., Weber, B.: Opening the Black Box of Team Processes and Emergent States: A Literature Review and Agenda for Research on Team Facilitation. *2014 47th Hawaii International Conference on System Sciences*. 473–482 (2014)
29. Tan, B., Wei, K.-K., Lee-Partridge, J.: Effects of Facilitation and Leadership on Meeting Outcomes in a Group Support System Environment. *European Journal of Information Systems*. 8(4), 233-246 (1999)
30. Sweller, J.: Cognitive Load during Problem Solving. *Cognitive Science*. 12(2), 257–285 (1988)
31. Paas, F., Renkl, A., Sweller, J.: Cognitive Load Theory: Instructional Implications of the Interaction between Information Structures and Cognitive Architecture. *Instructional Science*. 32(1/2), 1–8 (2004)
32. Saneiro, M., Santos, O.C., Salmeron-Majadas, S., and Boticario J.G.: Towards Emotion Detection in Educational Scenarios from Facial Expressions and Body Movements through Multimodal Approaches. *The Scientific World Journal*. Vol. 2014 (2014)

Scalable Dynamic Business Process Discovery with the Constructs Competition Miner

David Redlich^{1,2}, Thomas Molka^{1,3}, Wasif Gilani¹, Gordon Blair², and Awais Rashid²

¹ SAP Research Center Belfast, United Kingdom,
[david.redlich|thomas.molka|wasif.gilani]@sap.com

² Lancaster University, United Kingdom,
[gordon|marash]@comp.lancs.ac.uk

³ University of Manchester, United Kingdom

Abstract. Since the environment for businesses is becoming more competitive by the day, business organizations have to be more adaptive to environmental changes and are constantly in a process of optimization. Fundamental parts of these organizations are their business processes. Discovering and understanding the actual execution flow of the processes deployed in organizations is an important enabler for the management, analysis, and optimization of both, the processes and the business. This has become increasingly difficult since business processes are now often dynamically changing and may produce hundreds of events per second. The basis for this paper is the Constructs Competition Miner (CCM): A divide-and-conquer algorithm which discovers block-structured processes from event logs possibly consisting of exceptional behaviour. In this paper we propose a set of modifications for the CCM to enable scalable dynamic business process discovery of a run-time process model from a stream of events. We describe the different modifications and carry out an evaluation, investigating the behaviour of the algorithm on event streams of dynamically changing processes.

Key words: run-time models, business process management, process mining, complex event processing, event streaming, big data

1 Introduction

The success of modern organizations has become increasingly dependent on the efficiency and performance of their employed business processes (BPs). These processes dictate the execution order of singular tasks to achieve certain business goals and hence represent fundamental parts of most organizations. In the context of business process management, the recent emergence of Big Data yields new challenges, e.g. more analytical possibilities but also additional run-time constraints. An important discipline in this area is Process Discovery: It is concerned with deriving process-related information from event logs and, thus, enabling the business analyst to extract and understand the actual behaviour of a business process. Even though they are now increasingly used in commercial settings, many of the developed process discovery algorithms were designed to work in a static fashion, e.g. as provided by the ProM framework [15], but are

not easily applicable for processing real-time event streams. Additionally, the emergence of Big Data results in a new set of challenges for process discovery on event streams, for instance [11, 16]: (1) *diversity of event formats* from different sources, (2) *high event frequency* (e.g. thousands of events per second), and (3) *less rigid processes* (e.g. BPs found on the operational level of e-Health and security use-cases are usually subjected to frequent changes).

With the focus on addressing the latter two of these challenges, we propose in this paper modifications for the Constructs Competition Miner (CCM) [10] to enable Scalable Dynamic Process Discovery as proposed in [11]. The CCM is a process discovery algorithm that follows a divide-and-conquer approach to directly mine a block-structured process model which consists of common BP-domain constructs and represents the main behaviour of the process. This is achieved by calculating global relations between activities and letting different *constructs* compete with each other for the most suitable solution from top to bottom using "soft" constraints and behaviour approximations. The CCM was designed to deal with noise and not-supported behaviour. To apply the CCM on event streams the algorithm was split up into two individually operating parts:

1. **Run-time footprint calculation**, i.e. the current footprint¹, which represents the abstract "state" of the system, is updated with occurrence of each event. Since every occurring event constitutes a system state transition, the algorithmic execution-time needs to be kept to a minimum.
2. **Scheduled footprint interpretation**, i.e. from the footprint the current business process is discovered in a scheduled, reoccurring fashion. Since this part is executed in a different lifecycle it has less execution-time constraints. In this step the abstract "computer-centric" footprint is transformed into a "human-centric" business process representation.

The remainder of this paper provides essential background information (Section 2), a discussion of related work (Section 3), a summarized description of the original CCM (Section 4), the modifications that were carried out on top of the CCM to enable Scalable Dynamic Process Discovery (Section 5), an evaluation of the behaviour of the resulting algorithm for event streams of dynamically changing processes (Section 6), and an outlook of future work (Section 7).

2 Background

Business Processes are an integral part of modern organizations, describing the set of activities that need to be performed, their order of execution, and the entities that execute them. Prominent BP examples are Order-to-Cash or Procure-to-Pay. According to Ko et al. BPs are defined as "*...a series or network of value-added activities, performed by their relevant roles or collaborators, to purposefully achieve the common business goal*" [4]. A BP is usually described by a *process model* conforming to a business process standard, e.g. Business Process Model and Notation (BPMN) [9], or Yet Another Workflow Language (YAWL) [13]. In this paper, we will focus on business processes consisting of a set of common

¹ footprint is a term used in the process discovery domain, abstractly representing existent "behaviour" of a log, e.g. activity "a" is followed by activity "b"

control-flow elements, supported by most of the existing BP standards: start and end events, activities (i.e. process steps), parallel gateways (AND-Split/Join), and exclusive gateways (XOR-Split/Join) (see [9, 13]). In Figure 1 an example process involving all the introduced elements is displayed. Formally, we define a business process model as follows [10]:

Definition 1 A business process model is a tuple $BP = (A, S, J, E_s, E_e, C)$ where A is a finite set of activities, S a finite set of splits, J a finite set of joins, E_s a finite set of start events, E_e a finite set of end events, and $C \subseteq F \times F$ the path connection relation, with $F = A \cup S \cup J \cup E_s \cup E_e$, such that

- $C = \{(c_1, c_2) \in F \times F \mid c_1 \neq c_2 \wedge c_1 \notin E_e \wedge c_2 \notin E_s\}$,
- $\forall a \in A \cup J \cup E_s : |\{(a, b) \in C \mid b \in F\}| = 1$,
- $\forall a \in A \cup S \cup E_e : |\{(b, a) \in C \mid b \in F\}| = 1$,
- $\forall a \in J : |\{(b, a) \in C \mid b \in F\}| \geq 2$,
- $\forall a \in S : |\{(a, b) \in C \mid b \in F\}| \geq 2$, and
- all elements $e \in F$ in the graph (F, C) are on a path from a start event $a \in E_s$ to an end event $b \in E_e$.

For a block-structured BP model it is furthermore required that the process is hierarchically organised [10], i.e. it consists of unique join-split-pairs, each representing either a single entry or a single exit point of a non-sequential BP construct, e.g. Choice, Parallel, Loop, etc. The example process in Figure 1 is a block-structured process. A similar representation gaining popularity in recent years is the *process tree*, as defined based on Petri nets/workflow nets in [5].

When a business process is automatically or semi-automatically executed with a BP execution engine, e.g. with a Business Process Management System (BPMS), an *event log* is produced, i.e. all occurred events are logged and stored. These logs and their contained events may capture different aspects of a process execution, e.g. a different granularity of events are logged. In this paper however, we only focus on a minimal set of event features: In order to allow the discovery of the control-flow, every event is required to have a reference (1) to the associated *process instance* and (2) to the corresponding *activity*. Furthermore, we assume that the log contains exactly one event for each activity execution, i.e. activity lifecycle events are not regarded. All events resulting from the execution of the same *process instance* are captured in one *trace*. A trace is assumed to be independent from other traces, i.e. the execution order of a process instance is not in any way dependent on the execution of a second instance. Accordingly, an event e is represented by a pair $e = (t, a)$ where $t \in \mathbf{N}$ is the unique identifier of the trace and $a \in A$ is a unique reference to the executed activity.

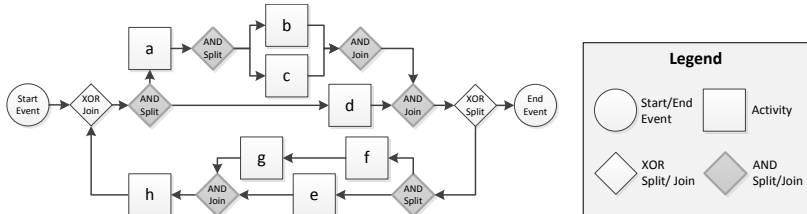


Fig. 1. Example business process with all element types included

The research area of *Process Discovery* is concerned with the extraction of a business process model from event logs without using any a-priori information [17]. Conventional challenges in process discovery originate from the motivation to achieve a high quality of results, i.e. discovered processes should support as accurately as possible the behaviour contained in the log. In particular that means, process discovery algorithms have to deal with multiple objectives, e.g. precision, simplicity, fitness - over-fitting vs. under-fitting (see [17]). Process discovery algorithms are usually assumed to be carried out in a static way as an "offline" method. This is reflected by the fact that the input for these algorithms is an entire log as conceptually shown by the following definition:

Definition 2 *Let the log $L_n = [e_0, e_1, \dots, e_n]$ be a sequence of $n+1$ events ordered by time of occurrence ($\forall i < j \wedge e_i, e_j \in L_n : \text{time}(e_i) \leq \text{time}(e_j)$) and BP_n be the business process model representing the behaviour in L_n , then process discovery is defined as a function that maps a log L_n to a process BP_n :*

$$\text{ProcessDiscovery} : [e_0, e_1, \dots, e_n] \Rightarrow BP_n$$

3 Related Work

A large number of process discovery algorithms exist, e.g. Inductive Miner [5], HeuristicsMiner [19], alpha-miner [14] and CCM [10]. These and many algorithms have in common that at first a *footprint* of the log is created based on which the process is constructed. Similar to the CCM, the following related algorithms also discover block-structured processes: (1) Genetic process discovery algorithms that restrict the search space to block-structured process models, e.g. [2]. However, these are non-deterministic and generally have a high execution time due to exponentially expanding search space. (2) Another relevant approach that is conceptually similar to the CCM is proposed in [5], the Inductive Miner (IM): A top-down approach is applied to discover block-structured Petri nets. The original algorithm evaluates constraints based on local relationships between activities in order to identify the representing construct in an inductive fashion. In recent work, the IM has also been extended to deal with noise [6]. Generally, in all discovery approaches based on footprints known to the authors the footprint is represented by a *direct neighbours* matrix representing information about the local relations between the activities, e.g. for the BP of Figure 1: *h* can only appear *directly* after *g* or *e*. As discussed in Section 4 the CCM on the other hand extracts the process from a footprint based on global relations between activities, e.g. *h* appears *at some point* after *g* or *e*.

However, of little importance for conventional process discovery algorithms is their practicality with regards to an application during run-time: as defined in Definition 2 process discovery is a static method that analyses an event log in its entirety. An alternative to this approach is the immediate processing of events when they occur to information of an higher abstraction level in order to enable a real-time analysis. This approach is called Complex Event Processing (CEP): a method that deals with the event-driven behaviour of large, distributed enterprise systems [7]. More specifically, in CEP events produced by the systems are captured, filtered, aggregated, and finally abstracted to complex events

representing high-level information about the situational status of the system, e.g. performance, control-flow, etc. The need for monitoring aspects of business processes at run-time by applying CEP methodologies has been identified by Ammon et al., thus coining the term Event-Driven Business Process Management (EDBPM) - a combination of two disciplines: Business Process Management (BPM) and Complex Event Processing [1]. The dynamic process discovery solution proposed in this paper is an application of EDBPM (see Section 5).

In the context of process discovery, an often used term for discovering processes from event streams is *Streaming Process Discovery*. In [3] the Heuristic-sMiner has been modified for this purpose by maintaining queues of fixed size $n \in \mathbb{N}$ containing the latest n events, i.e. the queues function as a "sliding window" over the event stream. Three different approaches of how to process these queues to a footprint have been proposed: (1) Stationary - every queue entry has the same weight, (2) Ageing - older entries have a decreasing weight, and (3) Self-Adapting Ageing - the factor with which the influence of older entries decreases is dependent on whether a concept drift² has been detected (quickly decreasing) or the process is assumed to be stationary (slowly decreasing). Additionally, Lossy Counting, a technique using approximate frequency count, has been investigated as a modification. A second approach for discovering concept drifts on event streams is presented in [8]: an incremental discovery of declarative process models using the stationary approach and Lossy Counting.

4 Static Constructs Competition Miner

The CCM as described in [10] is a deterministic process discovery algorithm that operates in a static fashion and follows a divide-and-conquer approach which, from a given event log, directly mines a block-structured process model that represents the main behaviour of the process. The CCM has the following main features [10]: (1) A deadlock-free, block-structured business process without duplicated activities is mined; (2) The following BP constructs are supported and can be discovered for single activities: Normal, Optional, Loopover, and Loopback; or for a set of activities: Choice, Sequence, Parallel, Loop, Loopover-Sequence, Loopover-Choice, Loopover-Parallel (see Figure 2), and additionally all of them as optional constructs - these are constructs supported by the majority of business process standards like BPMN or YAWL; (3) If conflicting or exceptional behaviour exists in the log, the CCM picks the "best" fitting BP construct.

Algorithm 1 shows the conceptual methodology of the CCM algorithm in pseudocode. The CCM applies the divide-and-conquer paradigm and is implemented in a recursive fashion (see lines 7, 16, and 17). At the beginning `getFootprintAndBuildConstruct` is initially called for all involved activities ($A_m = A$) with the process bp consisting of only a start and end element. The recursive function is first creating a footprint fp from the given log L only considering the activities specified in set A_m (at the beginning all involved activities). In a next step it will be decided which is the best construct to represent the behaviour captured by fp : (1) if the activity set A_m only consists of one element,

² A concept drift in this context is a behavioural change in the monitored process

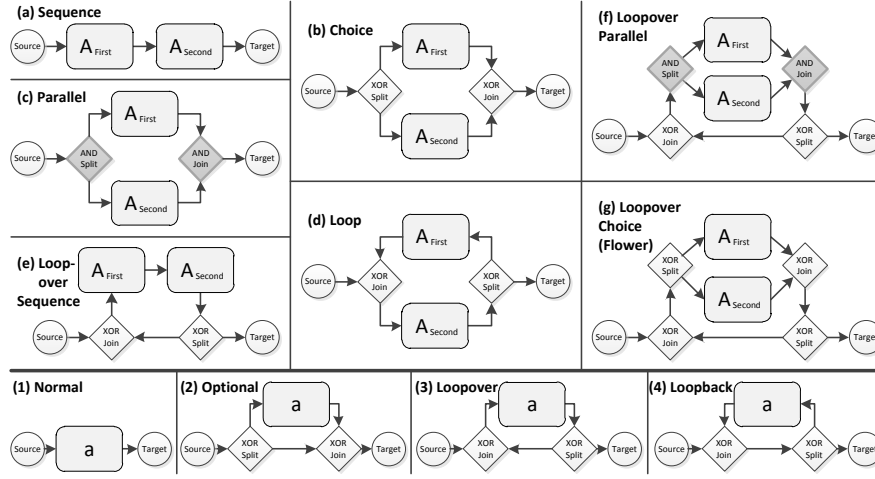


Fig. 2. Business Process Constructs Supported by the CCM [10]

Algorithm 1: Methodology of the CCM in Pseudocode

```

Data:  $\text{Log } L$ 
Result:  $BP \text{ } bp$ 
1 begin
2    $A \leftarrow \text{getSetOfAllActivitiesInLog}(L);$ 
3    $BP \text{ } bp \leftarrow \text{buildInitialBPWithStartAndEnd}();$ 
4    $bp \leftarrow \text{getFootprintAndBuildConstruct}(A, L, bp);$ 
5   return  $bp;$ 

6 Function  $\text{getFootprintAndBuildConstruct}(A_m, \text{Log } L, BP \text{ } bp)$ 
7    $\text{Footprint } fp = \text{extractFootprintForActivities}(A_m, L);$ 
8   if  $|A_m| = 1$  then
9      $\text{Construct } c \leftarrow \text{analyseConstructForSingleActivity}(fp);$ 
10     $bp \leftarrow \text{createSingleActivityConstruct}(c, A_m);$ 
11  else
12     $\text{ConstructsSuitability}[] \text{ } cs \leftarrow \text{calculateSuitabilityForConstructs}(fp, A_m);$ 
13     $(\text{Construct } c, A_{first}, A_{second}) \leftarrow \text{constructCompetition}(cs, A_m);$ 
14     $bp \leftarrow \text{createBlockConstruct}(c, bp);$ 
15     $bp \leftarrow \text{getFootprintAndBuildConstruct}(A_{first}, L, bp);$ 
16     $bp \leftarrow \text{getFootprintAndBuildConstruct}(A_{second}, L, bp);$ 
17  return  $bp;$ 

```

it will be decided which of the single activity constructs (see bottom of Figure 2) fits best - the process bp will then be enriched with the new single activity construct (see line 11); (2) If the activity set A_m contains more than one element, the suitability for each of the different constructs is calculated for any two activities $x, y \in A_m$ based on "soft" constraints and behaviour approximations, e.g. activities a and b are in a strong Sequence relationship. The result of this calculation (line 13) is a number of suitability matrices, one for each construct. In the subsequent competition algorithm it is determined what is the best combination of (A) the construct type $c \in \{Sequence, Choice, Loop, \dots\}$, and (B) the two subsets A_{first} and A_{second} of A_m with $A_{first} \cup A_{second} = A_m$, $A_{first} \cap A_{second} = \{\}$, and $A_{first}, A_{second} \neq \{\}$, that best accommodate all x, y -pair relations of the corresponding matrix of construct c (line 14). The construct is then created and

added to the existing process model bp (line 15), e.g. XOR-split and -join if the winning construct c was *Choice*. At this stage the recursive method calls will be executed to analyse and construct the respective behaviour for the subsets A_{first} and A_{second} . The split up of the set A_m continues in a recursive fashion until it cannot be divided any more, i.e. the set consists of a single activity (see case (1)). The process is completely constructed when the top recursive call returns.

Of particular interest for the transformation of the CCM algorithm to a solution for scalable dynamic process discovery is the composition of the footprint and its calculation from the log. As opposed to many other process discovery algorithms, e.g. alpha-miner [14], the footprint does not consist of absolute relations, e.g. h is followed by a (see example in Figure 1), but instead holds relative relation values, e.g. a is eventually followed by g in $0.4 \cong 40\%$ of the traces. Furthermore, the footprint only contains global relations between activities in order to guarantee a low polynomial execution time for the footprint interpretation [10]. The footprint of the CCM contains information about: (1) the occurrence of each involved activities $x \in A_m$, i.e. how many times x appears at least once per trace, how many times an x appears on average per trace, and how many times the trace started with x ; (2) the global relations of each activity pair $x, y \in A_m$, i.e. in how many traces x appears sometime before the *first* occurrence of y in the trace, and in how many traces x appears sometime before *any* occurrence of y in the trace³. All measures in the footprint are relative to the number of traces in the log. Furthermore, not only one overall footprint is created for the CCM but also for every subset A_{first} and A_{second} , that is created during execution, a new sub-footprint is created (see Algorithm 1).

5 Dynamic Constructs Competition Miner

As established in Section 1, increasingly dynamic processes and the need for immediate insight require current research in the domain of process mining to be driven by a set of additional challenges. To address these challenges the concept of Scalable Dynamic Process Discovery (SDPD), an interdisciplinary concept employing principles of CEP, Process Discovery, and EDBPM, has been introduced in [11]: "SDPD describes the method of monitoring one or more BPMSs in order to provide at any point in time a reasonably accurate representation of the current state of the processes deployed in the systems with regards to their control-flow, resource, and performance perspectives as well as the state of still open traces." That means, any potential changes in the mentioned aspects of the processes in the system that occur during run-time have to be recognized and reflected in the continuously updated "current state" of the process. Due to its purpose, for solutions of SDPD an additional set of requirements applies. For this paper, the most relevant of them are [11]:

- *Detection of Change*: An SDPD solution is required to detect change in two different levels defined in [12]: (1) Reflectivity: A change in a process instance

³ This stands in contrast to existing discovery solutions since in the CCM the footprint and its interpretation is not based on *local* relationships between activity occurrences, e.g. direct neighbours, but based on *global* relationships between them.

- (trace), i.e. every single event represents a change in the state of the associated trace. (2) Dynamism: A change on the business process level, e.g. because events/traces occurred that contradicts with the currently assumed process.
- *Scalability/Algorithmic Run-time*: An SDPD solution is applied as CEP concept and has to be able deal with large business processes operating with a high frequency, i.e. the actual run-time of the algorithms becomes very important. Additionally, the key algorithms are required to be scalable to cope with increasing workload at minimal possible additional computational cost.

Motivated by these challenges the initial process discovery approach was altered to allow for dynamic process discovery. As opposed to the traditional *static* methodology (see Definition 2), *dynamic* process discovery is an iterative approach as defined in the following:

Definition 3 Let $\log L_n = [e_0, e_1, \dots, e_n]$ be a sequence of $n+1$ events ordered by time of occurrence ($\forall i < j \wedge e_i, e_j \in L_n : \text{time}(e_i) \leq \text{time}(e_j)$) and BP_n be the business process model representing the behaviour in L_n , then dynamic process discovery is defined as a function that projects the tuple (e_n, BP_{n-1}) to BP_n :

$$\text{DynamicProcessDiscovery} : (e_n, BP_{n-1}) \Rightarrow BP_n$$

As described in Section 4, the CCM is a static mining algorithm and has to be modified in order to enable SDPD. The result of this modifications is called Dynamic CCM (DCCM). However, two restrictions for the DCCM with regards to the previously mentioned requirements of SDPD apply: (1) instead of discovering change on the BP perspectives control-flow, resources, and performance perspective, the DCCM described in this paper only focuses on discovering change in the control-flow, and (2) only change on the abstraction level of Dynamism is detected, i.e. whether or not the control-flow of the process has changed - the detection of change on the abstraction level of Reflectivity will not be supported by the DCCM. Additionally to the requirements of SDPD the DCCM features the following important aspects: (1) *robust*: if conflicting, exceptional, or not representable behaviour occurs in the event stream, the DCCM does not fail but always picks the BP construct that best accommodates the recorded behaviour; (2) *deterministic*: the DCCM yields the exact same output BP for the same input stream of events.

The following modifications were applied to the default CCM to create the DCCM and are described in more detail in the remainder of this section:

1. Splitting up the algorithm in two separate parts: one for dynamically updating the current footprint(s) complying to the scalability requirement, and one for interpreting the footprint into a BP model which has less restrictions with regards to its execution-time.
2. In the CCM the footprint is calculated in relation to all occurring traces. This is not applicable for SDPD since the number of traces should not have an influence on the execution-time of any component of an SDPD solution. For this reason the footprint has to be calculated in a dynamic fashion, i.e. an event-wise footprint update independent from the previously occurred number of events or traces.

3. The original behaviour of the CCM to carry out a footprint calculation for every subset that has been created by the divide-and-conquer approach is not optimal as then the DCCM would have to extract up to $2 * n + 1$ different footprints if only one activity was split-up from the main set for each recursion.⁴ This has been improved for the DCCM: for the most common constructs Choice and Sequence the sub-footprints are automatically derived from the parent footprint.
4. In rare cases it can happen that for every appearing event the state of the process is alternating between a number of different control-flows. This is caused by "footprint equivalent" BP models, i.e. two models are footprint equivalent if they both express the behaviour captured by the footprint. We introduce a measure which favours the last control-flow state in order to prevent the described behaviour.

5.1 Methodology of the Dynamic CCM

The original CCM algorithm had to be split up into two separate parts in order to comply to the scalability requirement of SDPD. A component triggered by the occurrence of a new event to update the dynamic footprint and a component decoupled from the event processing which interprets the footprint into a BP Model. The conceptual methodology of the DCCM is depicted in Figure 3. The components, models, and functionality of the DCCM are described in the following: Events from the monitored *Enterprise System*, in which the end-to-end process is deployed, are fed into an event stream. The *Footprint Update* component is the receiver of these events and processes them directly into changes on the overall *Dynamic Footprint* which represents the abstract state of the monitored business process. If additional footprints for subsets of activities are required as specified by the *Sub-Footprint Configurations*, e.g. if a Loop or Parallel construct was identified, then these sub-footprints are also updated (or created if they were not existent before). The *Dynamic Footprint(s)* can then at any point in time be compiled to a human-centric representation of the business process by the *Footprint Interpretation* component, i.e. the abstract footprint representation is interpreted into knowledge conforming to a block-structured BP model. In the DCCM this interpretation is scheduled dependent on how many new completed traces appeared, e.g. the footprint interpretation is executed once every 10 terminated traces. If the *interpretation frequency* $m \in \mathbb{N}$ of the DCCM is set to 1 a footprint interpretation is executed for every single trace that terminated. The *Footprint Interpretation* algorithm works similar to the CCM algorithm shown in Algorithm 1; but instead of extracting footprints from a log (line 8), the modified algorithm requests the readily available *Dynamic Footprint(s)*. If a sub-footprint is not yet available (e.g. at the beginning or if the process changed) the *Footprint Interpretation* specifies the request for a sub-footprint in the *Sub-Footprint Configurations* in the fashion of a feedback loop.

⁴ e.g. for $A = \{a, b, c, d\} : (a, b, c, d) \rightarrow ((a, b, c), (d)) \rightarrow (((a), (b, c)), (d)) \rightarrow (((a), ((b), (c))), (d))$, seven different footprints for sets $\{a, b, c, d\}$, $\{a, b, c\}$, $\{b, c\}$, $\{a\}$, $\{b\}$, $\{c\}$, $\{d\}$ need to be created - (,) denote the nested blocks that emerge while splitting the sets recursively.

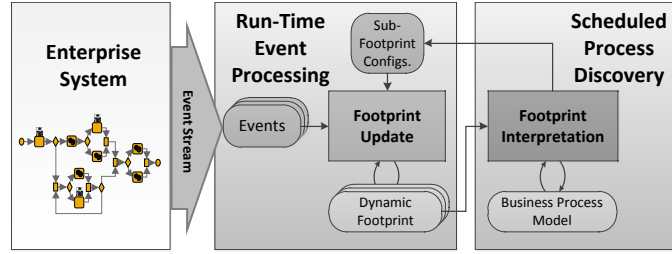


Fig. 3. Conceptual Methodology of the Dynamic CCM

Thus, *Sub-Footprint Configurations* and *Dynamic Footprints* act as interfaces between the two components, *Footprint Update* and *Footprint Interpretation*. The *Footprint Interpretation* cannot continue to analyse the subsets if no sub-footprint for these exist yet. In this case, usually occurring in the warm-up or transition phase, an intermediate BP model is created with activities containing all elements of the unresolved sets as depicted in Figure 4.

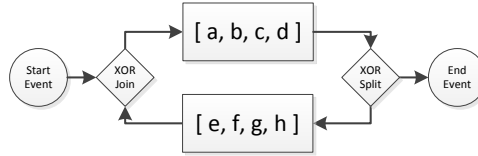


Fig. 4. Result of the *Footprint Interpretation* on an event stream produced by the example from Figure 1 if no sub-footprints for $\{a, b, c, d\}$ and $\{e, f, g, h\}$ are available yet - only the top-level loop has been discovered

5.2 Run-time Update of the Dynamic Footprint

The *Footprint Update* component processes events to changes in the *Dynamic Footprint*, i.e. updates the abstract representation of the process state. The original footprint extraction of the CCM algorithm calculates all values in relation to the number of occurred traces, i.e. every trace's influence on the footprint is equal: $\frac{1}{|traces|}$. To comply to the scalability requirement of SDPD the footprint update calculation should only take a fixed amount of time, independent from the total number of previously occurred events or traces. An increase of the total number of involved activities can cause, however, a linear increase of the execution-time due to the recalculation of the relations between the occurred activity and, in the worst case, all other activities. The independence from previous traces is the reason the footprint is calculated in a dynamic fashion, i.e. the dynamic footprint is incrementally updated in a way that older events "age" and thus have less influence than more recent events.

The ageing approach that is utilized in the *Footprint Update* of the DCCM is the creation of an individual *trace footprint*⁵ (*TFP*) for each trace and add it multiplied by the *trace influence factor* $t_{if} \in \mathbb{R}$ to the current dynamic overall footprint (*DFP*) multiplied by $1 - t_{if}$, e.g. for $t_{if} = 0.01$:

⁵ the occurrence values for activities as well as the global relations (see end of Section 4) are represented in the trace footprint by absolute statements *true* $\equiv 1$ if it occurred and *false* $\equiv 0$ if not

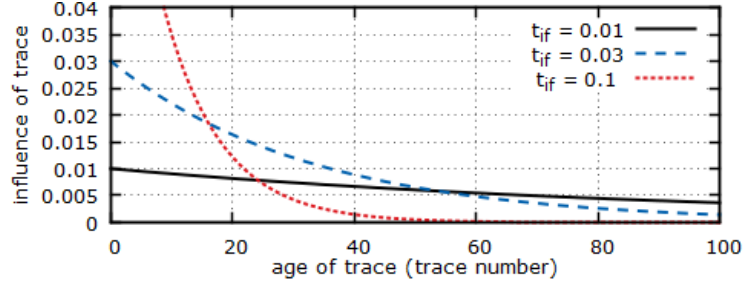


Fig. 5. Development of the influence of a trace for different trace influence factors(t_{if})

$DFP = 0.01 * TFP + 0.99 * DFP$. That means, a trace footprint TFP_i has at the beginning the influence of 0.01, after another TFP_{i+1} has been added the influence of TFP_i decreases to $0.01 * 0.99$, and after another $0.01 * 0.99^2$ and so on. By applying this incremental method, older TFP are losing influence in the overall dynamic footprint. Figure 5 shows how the influence of a trace is dependent on its "age": If $t_{if} = 0.1$, the influence of a trace that appeared 60 traces ago became almost irrelevant. At the same time if $t_{if} = 0.01$ the influence of a trace of the same age is still a little more than half of its initial influence when it first appeared. Essentially, the purpose of the *trace influence factor* t_{if} is to configure the "memory" and adaptation rate of the footprint update component.

Another important dynamism feature that had to be implemented was the possibility to add an activity that has not appeared before. A new activity is first recorded in the respective trace footprint. When the trace is terminated it will be added to the overall footprint in which it is not contained yet. The factored summation of both footprints to build the new dynamic footprint is carried out by assuming that a not previously in the dynamic overall footprint contained relation value is 0. An exception of this behaviour is the "warm-up" phase of the *Footprint Update*, i.e. if the amount of occurred traces is $< \frac{1}{t_{if}}$ then the influence of the dynamic footprint is $\frac{|traces|}{|traces|+1}$ and of the trace footprint $1 - \frac{|traces|}{|traces|+1}$. For instance if $t_{if} = 0.01$ and $|traces| = 9$ then is a new dynamic footprint calculated with $DFP_{10} = \frac{1}{10} * TFP + \frac{9}{10} * DFP_9$ and for the next trace $DFP_{11} = \frac{1}{11} * TFP + \frac{10}{11} * DFP_9$. Because of this implementation the "warm-up" phase of the *Footprint Update* could be drastically reduced, i.e. processes were already completely discovered a few traces after the start of the monitoring.

Furthermore, activities that do not appear any more during operation should be removed from the dynamic footprint. This was implemented in the DCCM in the following way: If the *occurrence once* value of an activity drops below a removal threshold $t_r \in \mathbb{R}$, $t_r < t_{if}$ it will be removed from the dynamic footprint, i.e. all values and relations to other activities are discarded.

The fact that especially many Choice and Sequence constructs are present in common business processes, motivates an automated sub-footprint creation in the *Footprint Interpretation* based on the parent footprint rather than creating the sub-footprint from the event stream. This step helps to decrease the execution-time of the *Footprint Update* and was achieved by introducing an ex-

tra relation to the footprint⁶ - the *direct neighbours* relation as used by other mining algorithms (see Section 3). In the *Footprint Interpretation* this relation is then used for creating the respective sub-footprints for Sequence and Choice constructs but not for identifying BP constructs since the *direct neighbours* relation does not represent a global relation between activities.

5.3 Modifications in the Footprint Interpretation Component

As analysed in the beginning of this section, the original behaviour of the CCM to retrieve a sub-footprint for each subset that has been created by the divide-and-conquer approach is not optimal. This is why, in the *Footprint Interpretation* the DCCM calculates the sub-footprints for the most common constructs, Choice and Sequence, from the available parent footprint: (1) For the Choice construct the probability of the exclusive paths are calculated with $p_{first} = \sum_{x \in A_{first}} Fel(x)$ and $p_{second} = \sum_{x \in A_{second}} Fel(x)$ with $Fel(x)$ being the occurrences of x as first element (see CCM footprint description in Section 4). Then the relevant values of the parent footprint are copied into their respective new sub-footprints and normalized, i.e. multiplied with $\frac{1}{p_{first}}$ and $\frac{1}{p_{second}}$, respectively. (2) The sub-footprints for the Sequence construct are similarly built, but without the normalization. Instead, the direct neighbours relation, now also part of the dynamic footprint, is used to calculate the new overall probabilities of the sub-footprints.

If two or more BP constructs are almost identically suitable for one and the same footprint, a slight change of the dynamic footprint might result in a differently discovered BP. This may cause an alternating behaviour for the footprint interpretation, i.e. with almost every footprint update the result of the interpretation changes. This is undesirable behaviour which is why the competition algorithm was additionally modified as follows: All combinations of BP construct and subsets are by default penalized by a very small value, e.g. $\frac{t_{if}}{10}$, with the exception of the combination corresponding to the previously discovered BP model, hence reducing the risk of discovering alternating BP models.

6 Evaluation

The static CCM algorithm has been tested for its accuracy in [10]: (1) in a qualitative analysis the CCM was able to rediscover 64 out of 67 processes for which a log was produced through simulation. (2) in the second part of the evaluation the discovery performance of the CCM was compared to the mining algorithms HeuristicsMiner (HM) [19], Inductive Miner (IM) [6], and the Flower Miner (FM), all of which are readily available in the ProM nightly build [15]. For ten given logs (including real-life logs and publicly available logs) the results of the algorithms (each configured with their default parameters) were evaluated for their *trace fitness* f_{tf} , *precision* f_{pr} , *generalization* f_g , and *simplicity* f_s with the help of the *PNetReplayer* plugin [18]. The averaged results of this analysis are shown in Table 1; Note, that a lower simplicity value is better.

⁶ In rare cases (if Loop and Parallel constructs dominate) this modification can have a negative effect on the execution-time since extra information needs to be extracted without the benefit of mining less sub-footprints

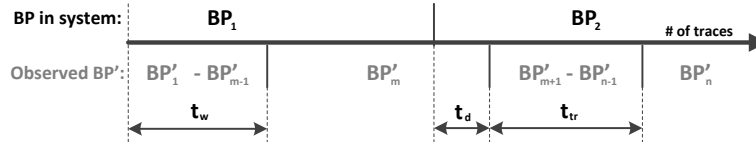
Table 1. Conformance results of the different discovery algorithms

Trace Fitness f_{tf}				Precision f_{pr}				Generalization f_g				Simplicity f_s			
HM	IM	FM	CCM	HM	IM	FM	CCM	HM	IM	FM	CCM	HM	IM	FM	CCM
0.919	0.966	1.0	0.979	0.718	0.622	0.124	0.663	0.941	0.915	0.992	0.930	155.3	122.8	56.4	111.9

In the remainder of this section early evaluation results of the DCCM are presented with regards to its capability of detecting certain basic changes of a real-time monitored business process. The basis of this evaluation is the example model in Figure 1 which is simulated and the resulting event stream fed into the DCCM. The CCM core is again configured with its default noise parameters. Figure 6 shows the different values we want to measure. In the figure BP_1 and BP_2 are the business processes deployed in the monitored system and BP'_1 to BP'_n are the discovered models by DCCM. Additionally, BP_1 and BP'_m are equivalent ($BP_1 \equiv BP'_m$) as well as BP_2 and BP'_n ($BP_2 \equiv BP'_n$). For this part of the evaluation the following measures are of interest:

- Warm-up: $t_w \in \mathbb{N}$ the amount of completed traces the DCCM needs as input at the start until the resulting model equivalently represents the process in the system, i.e. until $BP_1 \equiv BP'_m$.
- Change Detection: $t_d \in \mathbb{N}$ the amount of completed traces it takes to detect a certain change in the monitored process - from the point at which the process changed in the system to the point at which a different process was detected. When the change is detected the newly discovered process is usually not equivalent to the new process in the system BP_2 but instead represents parts of the behaviour of both processes, BP_1 and BP_2 .
- Change Transition Period: $t_{tr} \in \mathbb{N}$ the amount of completed traces it takes to re-detect a changed process - from the point at which the process change was detected to the point at which the correct process representation was identified, i.e. until $BP_2 \equiv BP'_n$. In this period multiple different business processes may be detected, each best representing the dynamic footprint at the respective point in time.

The first test will evaluate how the DCCM behaves at the beginning when first exposed to the event stream, more particularly, we want to determine t_w . Figure 7 shows a selection of the first few bp models extracted with *trace influence factor* $t_{if} = 0.01$ (see Section 5.2) and *interpretation frequency* $m = 10$, i.e. an interpretation is executed every 10 completed traces: After the first trace the discovered process is a sequence reflecting the single trace that defines the process at that point in time. At trace 10, which is the next scheduled footprint interpretation, the the algorithm discovered a Loop construct but cannot further analyse the subsets since the corresponding sub-footprint was not requested yet. Because of that, the feedback mechanism via the *Sub-Footprint Configurations* is utilized by the *Footprint Interpretation* algorithm to register the creation of

**Fig. 6.** Measures for Detection of BP Change in System

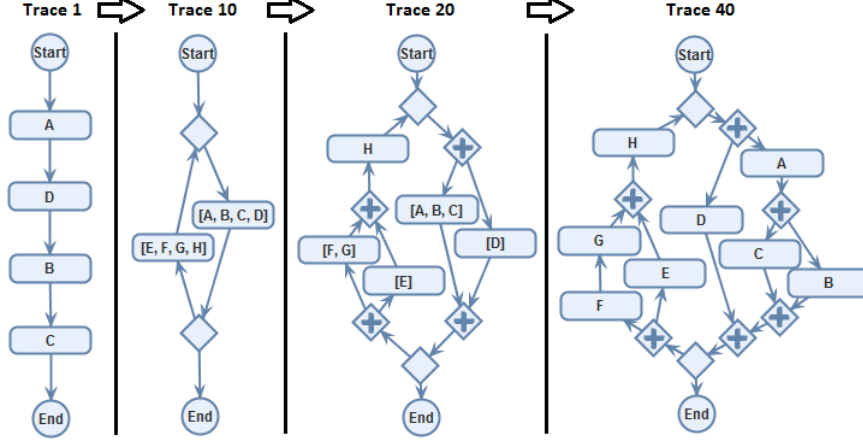


Fig. 7. The Evolution of the Discovered BP Model During the Warm-up Phase

the missing sub-footprints. In the next scheduled run of the footprint interpretation, the Parallel construct of a, b, c , and d is discovered but again the analysis can not advance since a sub-footprint for the individual activity subsets has not been created yet. Activities e, f, g , and h seem to have appeared only in exactly this sequence until trace 20. Skipping one of the interpretation steps, we can see that at trace 40 the complete process has been mined, i.e. $t_w = 40$.

In Figure 8 the development of t_w for different $m \in \{1, 2, 3, 6, 10\}$ and $t_{if} \in \{0.001, 0.005, 0.01, 0.03\}$ is depicted. The warm-up phase seems generally very short and not strongly influenced by t_{if} . For $m = 10$ the warm-up phase cannot be any shorter because the example process consists of a block-depth of 3: Parallel-in-Parallel-in-Loop, i.e. 3 subsequent requests for sub-footprints have to be made. This is an indicator that the modification effort to shorten the warm-up phase had a positive effect. A small decrease of t_w can be noticed when increasing the *trace influence factor* t_{if} for small m , e.g. $m \in \{1, 2, 3\}$.

In a second test we applied a change to the business process in the monitored system and are interested in the behaviour of the DCCM as well as in the change detection t_d and the change transition period t_{tr} . Figure 9 shows the evolution of the discovered BP model with *trace influence factor* $t_{if} = 0.01$ and *interpretation frequency* $m = 10$. The change applied is the move of activity a from the position before the inner Parallel construct to the position behind it (see traces 5750 and 6310). The change was applied after 5753 traces. The footprint interpretation

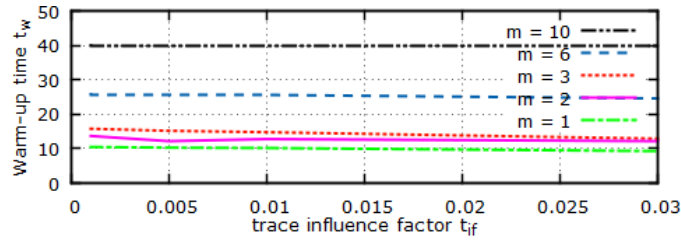


Fig. 8. The Warm-up Time in Relation to the Trace Influence Factor

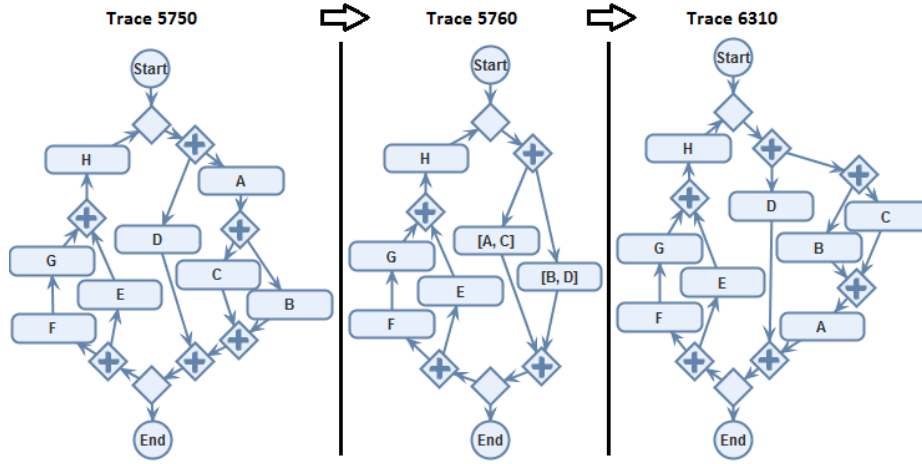


Fig. 9. The Evolution of the Discovered BP Model During a Change (Move of A)

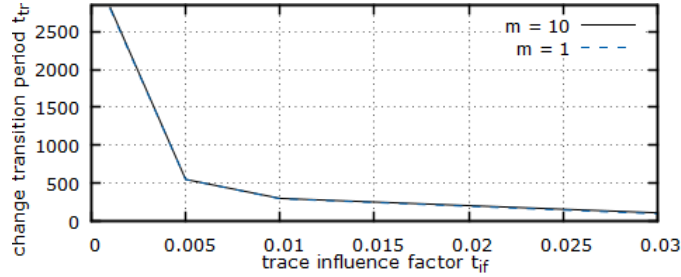


Fig. 10. The Change Transition Period in Relation to the Trace Influence Factor

detects at the first chance to discover the change (trace 5760) a concept drift and finds via competition the best fitting construct: Parallel of a, c and b, d . The change detection t_d seemed to be independent from m and t_{if} and was in all cases immediately recognized⁷. In Figure 10 the development of t_{tr} for different $m \in \{1, 10\}$ and $t_{if} \in \{0.001, 0.005, 0.01, 0.03\}$ is shown. The change transition period t_{tr} was strongly influenced by t_{if} . If the value was very small ($t_{if} = 0.001$) a change took up to almost 5000 traces in order to be reflected correctly in the discovered BP model. On the other hand if the trace influence factor is chosen too high, e.g. $t_{if} = 0.05$, not all variations of the process are included in the dynamic footprint which results in frequently changing/alternating discovered BP models. This is more likely to occur in large business processes containing rarely executed but still relevant behaviour.

Additionally, first performance tests have been carried out for large artificially produced processes (without change). For a randomly created and strongly nested process consisting of 100 activities the throughput of the footprint update was close to 100,000 events per second and the footprint interpretation successfully discovered the process in a matter of seconds. Although not tested yet in a

⁷ Note, that other changes like deletion of an activity will take longer to recognise, since their existence still "linger" in the footprints "memory" for some time.

real-life setting, the shown results indicate that the DCCM is very suitable for discovering and monitoring large enterprise processes.

7 Conclusion and Future Work

In this paper we suggested modifications for the Constructs Competition Miner to enable Scalable Dynamic Process Discovery as proposed in [11]. The CCM is a process discovery algorithm that follows a divide-and-conquer approach to directly mine a block-structured process model which consists of common BP-domain constructs and represents the main behaviour of the process. This is achieved by calculating global relations between activities and letting the different supported constructs compete with each other for the most suitable solution from top to bottom using "soft" constraints and behaviour approximations. The CCM was designed to deal with noise and not-supported behaviour. To apply the CCM in a real-time environment it was split up into two separate parts, executed on different occasions: (1) the *footprint update* which is called for every occurring event and updates the dynamic footprint(s) and (2) the footprint interpretation which derives the BP model from the dynamic footprint through applying a modified top-down competition approach of the original CCM algorithm. The modifications on the CCM were mostly motivated by the scalability requirement of SDPD and successfully implemented which is shown by the performance results in the evaluation section. It was furthermore shown that changes in the monitored process are almost instantly detected.

The presented approach of Dynamic CCM (DCCM) is driven by the requirements of real life industrial use cases provided by business partners within the EU funded project TIMBUS. During the evaluation in the context of the use-cases it became apparent that this concept still has a number of limitations which are considered to be future work: (1) Changes in the state of the business process are usually detected almost immediately but it may take a long time until the new state of the system is reflected appropriately in the extracted business process model. This behaviour originates from the fact that the footprint and the interpreted business process are in a sort of intermediate state for a while until the influence of the old version of the business process has disappeared. Furthermore, the trace influence factor t_{if} is a pre-specified value but in reality it is dependent on how many traces we need to regard to represent all the "behaviour" of the model⁸. This in turn is strongly dependent on the amount of activities in the model, since more activities usually mean more control-flow behaviour. A possible future modification could be to have the influence factor dynamically adapt, i.e. similar to the self-adapting ageing proposed in [3]. (2) If no sub-footprint is available for a set of activities, the footprint interpreter does not further analyse this set. Through approximations or the use of the direct neighbours relation at least a "close enough" control-flow for the subset could be retrieved. (3) The discovery of the state of a business process should also comprise information of other perspectives than the control-flow, e.g. resource and performance.

⁸ if t_{if} is set too high normal behaviour unintentionally becomes exceptional behaviour

References

1. von Ammon, R., Ertlmaier, T., Etzion, O., Kofman, A., Paulus, T.: Integrating Complex Events for Collaborating and Dynamically Changing Business Processes. In: ICSOC/ServiceWave 2009 Workshops. LNCS, pp. 370–384. Springer, 2010
2. Buijs, J., Van Dongen, B., Van Der Aalst, W.: A genetic algorithm for discovering process trees. In: Evolutionary Computation (CEC). pp. 1-8, IEEE, 2012
3. Burattin, A., Sperduti, A., Van Der Aalst, W.: Heuristics Miners for Streaming Event Data. In: CoRR abs/1212.6383, 2012
4. Ko, Ryan K. L.: A computer scientist’s introductory guide to business process management (BPM), In: Crossroads Journal, ACM, 2009
5. Leemans, S., Fahland, D., Van Der Aalst, W.: Discovering Block-Structured Process Models from Event Logs - A Constructive Approach. In: Application and Theory of Petri Nets and Concurrency, LNCS, pp. 311–329, Springer, 2013
6. Leemans, S., Fahland, D., Van Der Aalst, W.: Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour, In: Business Process Management Workshops 2013, LNBIP, pp. 66–78, Springer, 2013
7. Luckham, D.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Professional, Reading, 2002
8. Maggi, F. M., Burattin, A., Cimitile, M., Sperduti, A.: Online Process Discovery to Detect Concept Drifts in LTL-Based Declarative Process Models, OTM 2013, LNCS, pp. 94–111, Springer, 2013
9. OMG Inc: Business Process Model and Notation (BPMN) Specification 2.0, <http://www.omg.org/spec/BPMN/2.0/PDF>. formal/2011-01-03, 2011
10. Redlich, D., Molka, T., Rashid, A., Blair, G., Gilani, W.: Constructs Competition Miner: Process Control-flow Discovery of BP-domain Constructs. In: 12th Int. Conf. on Business Process Management, LNCS, pp. 134–150, Springer, 2014
11. Redlich, D., Gilani, W., Molka, T., Drobek, M., Rashid, A., Blair, G.: Introducing a Framework for Scalable Dynamic Process Discovery. In: 4th Enterprise Engineering Working Conference (EEWC), LNBIP 174, pp. 151–166. Springer, 2014
12. Redlich, D., Blair, G., Rashid, A., Molka, T., Gilani, W.: Research Challenges for Business Process Models at Run-time. In: LNCS State-of-the-Art Survey Volume on Models@run.time, 2014
13. Van Der Aalst, W., Ter Hofstede, A.: YAWL: Yet Another Workflow Language, 2003
14. Van Der Aalst, W., Weijters, A., Maruster, L.: Workflow Mining: Discovering Process Models from Event Logs. IEEE Transactions on Knowledge and Data Engineering. 16(9):1128-1142, 2004
15. Van Der Aalst, W., Van Dongen, B.: *ProM : The Process Mining Toolkit*. Industrial Engineering. 489: 1-4, 2009
16. Van Der Aalst et al., *Process Mining Manifesto*. BPM 2011 Int. Workshops, 2011
17. Van Der Aalst, W.: Process Mining - Discovery, Conformance and Enhancement of Business Processes, Springer, 2011
18. Van Der Aalst, W., Adriansyah, A., Van Dongen, B.: Replaying history on process models for conformance checking and performance analysis. WIREs Data Mining and Knowledge Discovery, 2(2), 182-192, 2012
19. Weijters, A., Van Der Aalst, W., Alves de Medeiros, A.: Process Mining with the Heuristics Miner-algorithm. BETA Working Paper Series, WP 166, Eindhoven University of Technology, 2006.

Project partially funded by the European Commission under the 7th Framework Programme for research and technological development and demonstration activities under grant agreement 269940, TIMBUS project (<http://timbusproject.net/>).

Scalable Process Monitoring Through Rules and Neural Networks

Alan Perotti¹, Guido Boella¹, and Artur d’Avila Garcez²

¹ University of Turin, Italy

² City University London, United Kingdom

Abstract. In this paper we introduce RuleRunner, a Runtime Verification system for monitoring LTL properties over finite traces. By exploiting results from the Neural-Symbolic Integration area, a RuleRunner monitor can be encoded in a recurrent neural network. The results show that neural networks can perform real-time runtime verification and techniques of parallel computing can be applied to improve the performance in terms of scalability. Furthermore, our framework allows for property adaptation by using a standard neural network learning algorithm.

1 Introduction

A trend in high-performance computation is parallel computing, and the field of neural networks has been showing impressive improvements in performance, especially with the use of GPU accelerators. For example, in 2012 Google used approximately 1,000 CPU-based servers, or 16,000 CPU cores, to develop its neural network (1.7 billion parameters), which taught itself to recognise cats in a series of YouTube videos [13]. One year later, the Stanford team created an equally large network with only three servers using NVIDIA GPUs to accelerate the processing of the big data generated by the network [7]. Our first Research Question is the following:

Is it possible to use the connectionist paradigm and parallel computing techniques in order to improve the performance of a runtime verification system?

Neural networks are also known to offer intrinsic learning capabilities [11], and many fields using verification also involve learning-based tasks. For instance, Business Process (BP) Management [19] attempts to continuously improve business effectiveness and efficiency, typically relying on Business Process Management Systems (BPMS). A BPMS allows to automatically execute a BP according to its process schema, i.e., to a formalised model in which the actions to be performed and the control flow relations to be respected among them are specified. However, BP optimisation may ask the enterprise to be able to flexibly change and adapt such a predefined process schema, in response to expected situations (e.g., new laws, reengineering efforts, *concept drift* [19]) as well as to unanticipated exceptions and problems in the operating environment (e.g., emergencies) [12]. Our second Research Question is therefore:

Can one develop a single framework that combines efficient monitoring, possibly through parallel computing techniques, with property adaptation through learning?

In this paper, we answer both Research Questions positively. We present our runtime

monitoring system: RuleRunner creates and maintains a single, detailed state, avoiding the *possible worlds* branching structure typical of tableaux-based formulations. Furthermore, RuleRunner is designed as a set of rules that can be fired in parallel, rather than in a strict sequence. We show how a RuleRunner system can be translated into an equivalent logic program, in order to exploit results from the Neural-Symbolic Integration area [8] to encode it in a recurrent neural network. We validate our framework by comparing three implementations based on neural networks, observing how sparse form and GPU computing improve the performance, respectively, in terms of absolute speed and scalability. We show how in our framework learning corresponds to adapt the encoded property according to the observed behaviour of a system, but the application of learning strategies and approaches is outside the scope of this paper.

As an example, suppose several violations (signalled by an access control monitor) coming from a given area of a LAN turn out to be false positives [6]. The reasonable reaction is to relax the encoded property in order to correctly identify the transactions from the given area as acceptable. In this way, the security system still encodes the given property, but reducing the number of false positives makes the system more efficient. This paper is structured as follows: Section 2 introduces background and related work and Section 3 analyses our rule-based system, RuleRunner. Section 4 discusses the translation of our rule system in a neural network; Section 5 shows experimental results and a motivational example for parallel monitoring and property adaptation; Section 6 ends the paper with closing remarks and directions for future work.

2 Background and Related Work

2.1 Runtime Verification

Runtime Verification (RV) relates an observed system with a formal property ϕ specifying some desired behaviour. An RV module, or monitor, is defined as a device that reads a trace and yields a certain verdict [14]. A trace is a sequence of cells, which in turn are lists of observations occurring in a given discrete span of time. Runtime verification may work on finite (terminated), finite but continuously expanding, or on prefixes of infinite traces. While LTL is a standard semantic for infinite traces [17], there are many semantics for finite traces: FLTL [15], RVLTL [3], LTL3 [4], LTL \pm [10] just to name some. Since the standard LTL semantics is based on infinite behaviours, the issue is to close the gap between properties specifying infinite behaviours and finite traces. In particular, FLTL differs from LTL as it offers two *next* operators (X , \bar{X} in [3], X , W in this paper), called respectively *strong* and *weak* next. Intuitively, the strong (and standard) X operator is used to express with $X\phi$ that a next state must exist and that this next state has to satisfy property ϕ . In contrast, the weak W operator in $W\phi$ says that if there is a next state, then this next state has to satisfy the property ϕ . More formally, let $u = a_0..a_{n-1}$ denote a finite trace of length n . The truth value of an FLTL formula ψ (either $X\phi$ or $W\phi$) w.r.t. u at position $i < n$, denoted by $[u, i \models \psi]$, is an element of \mathbb{B} and is defined as follows:

$$[u, i \models X\phi] = \begin{cases} [u, i+1 \models \phi], & \text{if } i+1 < n \\ \perp, & \text{otherwise} \end{cases} \quad [u, i \models W\phi] = \begin{cases} [u, i+1 \models \phi], & \text{if } i+1 < n \\ \top, & \text{otherwise} \end{cases}$$

While RVLTL and LTL3 have been proven to hold interesting properties w.r.t. FLTL (see [3]), we selected FLTL as we think it captures a more intuitive semantics when

dealing with finite traces. Suppose to monitor $\phi = \Box a$ over a trace t , where a is observed in all cells: we have that $[t \models \phi]$ equals, respectively, \top in FLTL, $?$ in LTL3, and T^p in RVLTL. If t is seen as a prefix of a longer trace $t\sigma$, then LTL3 and RVLTL provide valuable information about how ϕ could be evaluated over σ . But if t is a conclusive, self-contained trace (e.g. a daily set of transactions), then the FLTL semantics captures the intuitive positive answer to the query *does a always hold in this trace?*

Several RV systems have been developed, and they can be clustered in three main approaches, based respectively on rewriting, automata and rules [14]. Within rule based approaches, RuleR [2] uses an original approach. It copes with the temporal dimension by introducing rules which may reactivate themselves in later stages of the reasoning. RuleRunner is inspired by this powerful idea, and in particular by the striking similarity of triggering rules and activating neurons. While various formalisms can be encoded in a RuleR monitor, we focus on FLTL and extend RuleR to allow structures of a fixed size that can be implemented in efficient networks. The next section will describe the difference in the two approaches in more detail.

2.2 Neural networks and Neural-symbolic Integration

An artificial Neural Network (NN, [11]) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. Within the umbrella term of *connectionist approach*, many NN models have been developed, varying from those with only one or two layers of single direction logic, to complicated models with multi-input and many-directional feedback loops and layers. On the whole, these systems use algorithms in their programming to determine control and organisation of their functions. What they do have in common, however, is the principle of non-linear, distributed, parallel and local processing and adaptation.

The main purpose of a neural-symbolic system is to bring together the connectionist and symbolic approaches exploiting the strengths of both paradigms and, hopefully, avoiding their drawbacks. In [18], Towell and Shavlik presented the influential neural-symbolic system KBANN (Knowledge-Based Artificial Neural Network), a system for rule insertion, refinement and extraction from feedforward neural networks. KBANN served as inspiration for the construction of the Connectionist Inductive Learning and Logic Programming (CILP) system [8] (example in Fig. 1). CILP's Translation Algorithm maps a general logic program P into a single-hidden-layer neural network N such that N computes the least fixed-point of P . In particular, rules are mapped onto hidden neurons, the preconditions of rules onto input neurons and the conclusion of the rules onto output neurons. The weights are then adjusted to express the dependence among all these elements. The obtained network implements a massively parallel model for Logic Programming, and it can perform inductive learning from examples, by means of standard learning strategies.

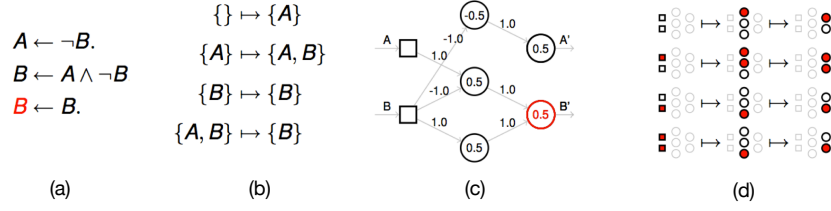


Fig. 1: Logic program P as a network N (CILP); (a) shows a simple logic program P . Applying CILP to P generates a NN N as depicted in (c). The input-outputs behaviour in P and N is shown in (b) and (d) respectively. For instance, if P is presented with the fact A , the first two clauses will produce A, B (second line in (b)). Similarly, if the input neuron representing A is activated in N , the signal propagation will activate the first two hidden neurons and both output neurons (second line in (d)).

3 The RuleRunner Rule System

RuleRunner is a rule-based online monitor observing finite but expanding traces and returning an FLTL verdict. RuleRunner accepts formulae ϕ generated by the grammar:

$$\phi ::= true \mid a \mid !a \mid \phi \vee \phi \mid \phi \wedge \phi \mid \phi U \phi \mid X\phi \mid W\phi \mid \Diamond\phi \mid \Box\phi \mid END$$

a is treated as an atom and corresponds to a single observation in the trace. We assume, without loss of generality, that temporal formulae are in negation normal form (NNF), e.g., negation operators pushed inwards to propositional literals and cancellations applied. W is the weak next operator. END is a special character that is added to the last cell of a trace to mark the end of the input stream.

Algorithm 1 RuleRunner monitoring (abstract)

```

1: function RR-MONITORING( $\phi$ , trace  $t$ )
2:   Build a monitor  $RR_\phi$  encoding  $\phi$ 
3:   while new cells exist in  $t$  do
4:     Observe the current cell
5:     Compute truth values of  $\phi$  in the current cell of  $t$ 
6:     if  $\phi$  is verified or falsified then
7:       return SUCCESS or FAILURE respectively
8:     end if
9:     Set up the monitor for the next cell in  $t$ 
10:  end while
11: end function

```

▷ Evaluation rules

▷ Reactivation rules

Given an FLTL formula ϕ and a trace t , Algorithm 1 provides an abstract description of the creation and runtime behaviour of a RuleRunner system monitoring ϕ over t . At first, a monitor encoding ϕ is computed. Second, the monitor enters the verification loop, composed by observing a new cell of the trace and computing the truth value of the property in the given cell. If the property is irrevocably satisfied or falsified in the current cell, RuleRunner outputs a binary verdict. If this is not the case (because the ϕ refers to cells ahead in the trace), the system shifts to the following cell and enters another monitoring iteration. The FLTL semantics guarantees that, if the trace ends, the verdict in the last cell of the trace is binary. RuleRunner is a runtime monitor, as it analyses one cell at a time and never needs to store past cells in memory nor peek future ones.

Definition 1. A RuleRunner system is a tuple $\langle R_E, R_R, S \rangle$, where R_E (evaluation rules) and R_R (reactivation rules) are rule sets, and S (for state) is a set of active rules, observations and truth evaluations.

Given a finite set of observations O and an FLTL formula ϕ over (a subset of) O , a state S is a set of observations ($o \in O$), rule names ($R[\psi]$) and truth evaluations ($[\psi]V$); $V \in \{T, F, ?\}$ is a truth value. A rule name $R[\psi]$ in S means that the logical formula ψ is under scrutiny, while a truth evaluation $[\psi]V$ means that the logical formula ψ currently has the truth value V . The third truth value, $?$ (*undecided*), means that it is impossible to give a binary verdict in the current cell.

Evaluation rules follow the pattern $R[\phi], [\psi^1]V, \dots, [\psi^n]V \rightarrow [\phi]V$ and their role is to compute the truth value of a formula ϕ under verification, given the truth values of its direct subformulae ψ^i (line 5 in Algorithm 1). For instance, $R[\Diamond\psi], [\psi]T \rightarrow [\Diamond\psi]T$ reads as *if $\Diamond a$ is being monitored and ψ holds, then $\Diamond\psi$ is true*.

Reactivation rules follow the pattern $[\phi]? \rightarrow R[\phi], R[\psi^1], \dots, R[\psi^n]$ and the meaning is that if one formula is evaluated to undecided, that formula (together with its subformulae) is scheduled to be monitored again in the next cell of the trace (line 9 in Algorithm 1). For instance, $[\Diamond\psi]? \rightarrow R[\Diamond\psi], R[\psi]$ means that *if $\Diamond\psi$ was not irrevocably verified nor falsified in the current cell of the trace, both ψ and $\Diamond\psi$ will be monitored again in the next cell*.

A RuleRunner feature is that rules never involve disjunctions. In RuleR, for instance, the simple formula $\Diamond a$ is mapped to the rule $R_{\Diamond a} : \longrightarrow a \mid R_{\Diamond a}$ and its meaning, intuitively, is that, if $\Diamond a$ has to be verified, either a is observed (thus satisfying the property) or the whole formula will be checked again (in the next cell of the trace). The same formula corresponds to the following set of rules in RuleRunner:

$$\begin{array}{ll} R[\Diamond a], [a]T \rightarrow [\Diamond a]T & R[\Diamond a], [a]?, END \rightarrow [\Diamond a]F \\ R[\Diamond a], [a]? \rightarrow [\Diamond a]? & [\Diamond a]? \rightarrow R[a], R[\Diamond a] \\ R[\Diamond a], [a]F \rightarrow [\Diamond a]? & \end{array}$$

The disjunction in the body of the RuleR rule corresponds to the additional constraints in the head of the RuleRunner rules. Therefore, where RuleR generates a set of alternative hypotheses and later matches them with actual observations, RuleRunner maintains a detailed state of exact information. This is achieved by means of evaluation tables: three-valued truth tables (as introduced by Lukasiewicz [16]) annotated with *qualifiers*. Each evaluation rule for ϕ corresponds to a single cell of the evaluation table for the main operator of ϕ ; a *qualifier* is a subscript letter providing additional information to $?$ truth values. Figure 2 gives the example for disjunction. Qualifiers (B, L, R in this case) are used to store and propagate detailed information about the verification status of formulae. For instance, if ϕ is undecided and ψ is false when monitoring $\phi \vee \psi$ (highlighted cell in Figure 2), $?_L$ means that the disjunction is undecided, but that its future verification state will depend on the truth value of the **L**eft disjunct. Note, in fact, how \vee_L is a unary operator. An example for this is monitoring $\Diamond b \vee a$ against a cell including only c : a is false, $\Diamond b$ is undecided (as b may be observed in the future), and the whole disjunction will be verified/falsified in the following cells depending on $\Diamond b$ only.

\vee	T	$?$	F
T	T	T	T
$?$	T	$?$	$?$
F	T	$?$	F

\vee_B	T	$?$	F
T	T	T	T
$?$	T	$?$	$?$
F	T	$?$	F

\vee_L	T	T
$?$	$?$	$?$
F	F	F

\vee_R	T	T
$?$	$?$	$?$
F	F	F

Fig. 2: Truth table (left) and evaluation tables (right) for \vee

\underline{a} (observation) $a \in \text{state}$ $\begin{bmatrix} T \\ F \end{bmatrix}$ $a \notin \text{state}$ $\begin{bmatrix} F \\ T \end{bmatrix}$	$\underline{!a}$ $a \in \text{state}$ $\begin{bmatrix} F \\ T \end{bmatrix}$ $a \notin \text{state}$ $\begin{bmatrix} T \\ F \end{bmatrix}$	\wedge_B $\begin{bmatrix} T & ? & F \\ T & T & ?_R & F \\ ? & ?_L & ?_B & F \\ F & F & F & F \end{bmatrix}$ \wedge_L $\begin{bmatrix} T \\ T \\ ? \\ F \end{bmatrix}$ \wedge_R $\begin{bmatrix} T \\ T \\ ? \\ F \end{bmatrix}$	\vee_B $\begin{bmatrix} T & ? & F \\ T & T & T & T \\ ? & T & ?_B & ?_L \\ F & T & ?_R & F \end{bmatrix}$ \vee_L $\begin{bmatrix} T \\ T \\ ? \\ F \end{bmatrix}$ \vee_R $\begin{bmatrix} T \\ T \\ ? \\ F \end{bmatrix}$
\square $\begin{bmatrix} T \\ ? \\ F \end{bmatrix}$ $\text{END} - \square$ $\begin{bmatrix} ?_K & T \\ ? & F \end{bmatrix}$	\diamond $\begin{bmatrix} T \\ ? \\ F \end{bmatrix}$ $\text{END} - \diamond$ $\begin{bmatrix} ? \\ F \end{bmatrix}$	W_M $\begin{bmatrix} T \\ ? \\ F \end{bmatrix}$ $\text{END} - W$ $\begin{bmatrix} ? \\ T \\ F \end{bmatrix}$	X_M $\begin{bmatrix} T \\ ? \\ F \end{bmatrix}$ $\text{END} - X$ $\begin{bmatrix} ? \\ F \\ F \end{bmatrix}$
U_A $\begin{bmatrix} T & ? & F \\ T & T & ?_A & ?_A \\ ? & T & ?_A & ?_B \\ F & T & ?_R & F \end{bmatrix}$	U_B $\begin{bmatrix} T & ? & F \\ T & T & ?_A & ?_A \\ ? & ?_L & ?_B & ?_B \\ F & F & F & F \end{bmatrix}$	U_L $\begin{bmatrix} T \\ T \\ ? \\ F \end{bmatrix}$	U_R $\begin{bmatrix} T \\ T \\ ? \\ F \end{bmatrix}$
$\text{END} - U$ $\begin{bmatrix} ?_A & F \\ ?_B & F \\ ?_L & F \\ ?_R & F \end{bmatrix}$			

Fig. 3: Complete set of evaluation tables

The complete set of evaluation tables is reported in Fig. 3, while the generation of evaluation and reactivation rules is summarised in Algorithm 2. The algorithm parses ϕ in a tree and visits the parsing tree in post-order. The system is built incrementally, starting from the system(s) returned by the recursive call(s). If ϕ is an observation (or its negation), an initial system is created, including two evaluation rules (as the observation may or may not occur), no reactivation rules and the single $R[\phi]$ as initial state. If ϕ is a conjunction or disjunction, the two systems of the subformulae are merged, and the conjunction/disjunction evaluation rules, reactivation rule and initial activation are added. The computations are the same if the main operator is U , but the reactivation rule will have to reactivate the monitoring of the two subformulae; in particular, U_A denotes the standard *until* operator, while U_B is the particular case where the ψ failed and the *until* operator cannot be trivially satisfied anymore. Formulae with X or W as main operator go through two phases: first, the formula is evaluated to undecided, as the truth value can't be computed until the next cell is accessed. Special evaluation rules force the truth value to false (for X) or true (for W) if no next cell exists. Then, at the next iteration, the reactivation rule triggers the subformula: this means that if $X\phi$ is monitored in cell i , ϕ is monitored in cell $i + 1$. ϕ is then monitored independently, and the $X\phi$ (or $W\phi$) rule enters a 'monitoring state' (suffix M in the table), simply mirroring ϕ truth value and self-reactivating. The evaluation of $\square\phi$ is false (undecided) when ϕ is false (undecided); it is also undecided when ϕ holds (as $\square\phi$ can never be true before the end of the trace),

but the K suffix indicates when, at the end of the trace, an undecided \square can be evaluated to true. Finally, $\Diamond\phi$ constantly reactivates itself and its subformula ϕ , unless ϕ is verified at runtime (causing $\Diamond\phi$ to hold), the trace ends ($\Diamond\phi$ fails).

Algorithm 2 Generation of rules

```

1: function INITIALISE( $\phi$ )
2:    $op \leftarrow$  main operator
3:   if  $op \in \{\square, \Diamond, X, W\}$  then
4:      $\langle R_E^1, R_R^1, S^1 \rangle \leftarrow$  Initialise( $\psi^1$ )
5:      $R_E \leftarrow R_E^1$ ;
6:      $R_R \leftarrow R_R^1$ ;
7:   else if  $op \in \{\vee, \wedge, U\}$  then
8:      $\langle R_E^1, R_R^1, S^1 \rangle \leftarrow$  Initialise( $\psi^1$ )
9:      $\langle R_E^2, R_R^2, S^2 \rangle \leftarrow$  Initialise( $\psi^2$ )
10:     $R_E \leftarrow R_E^1 \cup R_E^2$ ;  $R_R \leftarrow R_R^1 \cup R_R^2$ ;
11:   else
12:      $R_E \leftarrow \emptyset$ ;  $R_R \leftarrow \emptyset$ ;
13:   end if
14:    $Cells \leftarrow$  op's-evaluation-tables
15:   for all cell  $\in$  Cells do
16:     Convert cell to single rule  $r_e$ , substituting formula names
17:      $R_E \leftarrow R_E \cup r_e$ 
18:   end for
19:   if  $\phi$ -is-main-formula then
20:      $R_E \leftarrow R_E \cup ([\phi]T \rightarrow SUCCESS)$ 
21:      $R_E \leftarrow R_E \cup ([\phi]F \rightarrow FAILURE)$ 
22:      $R_E \leftarrow R_E \cup ([\phi]? \rightarrow REPEAT)$ 
23:   end if
24:   if  $op = a$  then  $S \leftarrow R[a]$ 
25:   else if  $op = !a$  then  $S \leftarrow R[!a]$ 
26:   else if  $op \in \{\vee, \wedge\}$  then  $S \leftarrow S^1 \cup S^2 \cup R[\phi]B$ 
27:   else if  $op = U$  then  $S \leftarrow S^1 \cup S^2 \cup R[\phi]A$ 
28:   else if  $op \in \{\square, \Diamond\}$  then  $S \leftarrow S^1 \cup R[\phi]$ 
29:   else if  $op \in \{X, W\}$  then  $S \leftarrow R[\phi]$ 
30:   end if
31:   if  $op \in \{\vee, \wedge\}$  then  $R_R \leftarrow R_R \cup ([\phi]?Z \rightarrow R[\phi]?Z)$ , for  $Z \in L, R, B$ 
32:   else if  $op = U$  then  $R_R \leftarrow R_R \cup ([\phi]?Z \rightarrow R[\phi]?Z, S^1, S^2)$ , for  $Z \in A, B, L, R$ 
33:   else if  $op \in \{\square, \Diamond\}$  then  $R_R \leftarrow R_R \cup ([\phi]? \rightarrow R[\phi], S^1)$ 
34:   else if  $op \in \{X, W\}$  then  $R_R \leftarrow R_R \cup ([\phi]? \rightarrow R[\phi]M, S^1) \cup ([\phi]?M \rightarrow R[\phi]M)$ 
35:   end if
36:   return  $\langle R_E, R_R, S \rangle$ 
37: end function

```

▷ Apply recursively to subformula(e)
 ▷ Compute and add evaluation rules for main operator
 ▷ Compute initial state for this subsystem
 ▷ Compute and add reactivation rules for main operator
 ▷ Return computed system

RuleRunner generates several rules for each operator, but this number is constant, as it corresponds to the size of evaluation tables plus special rules (like the SUCCESS one). The number of rules corresponding to $\phi \vee \psi$, for instance, does not depend in any way on the nature of ϕ or ψ , as only the final truth evaluation of the two subformulae is taken into account. The preprocessing phase creates the parse tree of the property to encode and adds a constant number of rules for each node (subformula). The obtained rule set does not change at runtime nor when monitoring new traces. During the runtime verification, for each cell of the trace the system goes through all rules exactly once. This is guaranteed by the post-order visit of the parsing tree, as shown in Algorithm 2, assuring pre-emption for rules evaluating simpler formulae. Therefore, the complexity of the system is inherently linear. This is not in contrast with known exponential lower bounds

for the temporal logic validity problem, as RuleRunner deals with the satisfiability of a property on a trace, thus tackling a different problem from the validity one (this distinction is also mentioned in [9]). A prototype for RuleRunner can be found [here](#).

As an example, consider the formula $\phi = a \vee \Diamond b$ and the trace $t = [c - a - b, d - b, END]$ (dashes separate cells and commas separate observations in the same cell). If monitoring ϕ over t , a fails in the first cell, while b is sought until the third cell, when it is observed. Thus the monitoring yields a success even before the end of the trace.

EVALUATION RULES

- $R[a], a \text{ is not observed} \rightarrow [a]F$
- $R[b], b \text{ is observed} \rightarrow [b]T$
- $R[b], b \text{ is not observed} \rightarrow [b]F$
- $R[\Diamond b], [b]T \rightarrow [\Diamond b]T$
- $R[\Diamond b], [b]F \rightarrow [\Diamond b]?$
- $R[a \vee \Diamond b]B, [a]F, [\Diamond b]? \rightarrow [a \vee \Diamond b]?_R$
- $R[a \vee \Diamond b]R, [\Diamond b]T \rightarrow [a \vee \Diamond b]T$
- $R[a \vee \Diamond b]R, [\Diamond b]? \rightarrow [a \vee \Diamond b]?_R$
- $[a \vee \Diamond b]T \rightarrow SUCCESS$

REACTIVATION RULES

- $[\Diamond b]? \rightarrow R[b], R[\Diamond b]$
- $[a \vee \Diamond b]?_R \rightarrow R[a \vee \Diamond b]R$

INITIAL STATE

- $R[a], R[b], R[\Diamond b], R[a \vee \Diamond b]B$

EVOLUTION OVER $[c - a - b, d - b, END]$

state	$R[a], R[b], R[\Diamond b], R[a \vee \Diamond b]B$
+ obs	$R[a], R[b], R[\Diamond b], R[a \vee \Diamond b]B, c$
eval	$[a]F, [b]F, [\Diamond b]?, [a \vee \Diamond b]?_R$
react	$R[b], R[\Diamond b], R[a \vee \Diamond b]R$
state	$R[b], R[\Diamond b], R[a \vee \Diamond b]R$
+ obs	$R[b], R[\Diamond b], R[a \vee \Diamond b]R, a$
eval	$[b]F, [\Diamond b]?, [a \vee \Diamond b]?_R$
react	$R[b], R[\Diamond b], R[a \vee \Diamond b]R$
state	$R[b], R[\Diamond b], R[a \vee \Diamond b]R$
+ obs	$R[b], R[\Diamond b], R[a \vee \Diamond b]R, b, d$
eval	$[b]T, [\Diamond b]T, [a \vee \Diamond b]T, SUCCESS$
STOP	PROPERTY SATISFIED

The behaviour of the runtime monitor is the following:

- At the beginning, the system monitors $a, b, \Diamond b$ and $a \vee \Diamond b$ (initial state = $R[a], R[b], R[\Diamond b], R[a \vee \Diamond b]B$). The $-B$ in $R[a \vee \Diamond b]B$ means that both disjuncts are being monitored.
- In the first cell, c is observed and added to the state S . Using the evaluation rules, new truth values are computed: a is false, b is false, $\Diamond b$ is undecided. The global formula is undecided, but since the trace continues the monitoring goes on. The $-R$ in $R[a \vee \Diamond b]R$ means that only the right disjunct is monitored: the system dropped a , since it could only be satisfied in the first cell.
- In the second cell, a is observed but ignored (the rules for its monitoring are not activated); since b is false again, $\Diamond b$ and $a \vee \Diamond b$ are still undecided.
- In the third cell, d is ignored but observing b satisfies, in cascade, $b, \Diamond b$ and $a \vee \Diamond b$. The monitoring stops, signalling a success. The rest of the trace is ignored.

4 Encoding the monitor in a Neural Network

The translation of a RuleRunner monitor in an equivalent neural network is composed by two main steps, respectively encoding a rule system in a logic program and the resulting logic program in a neural network. We hereby describe the details of the first step and explain how to use CILP for the second one. The first step of the neural encoding is the translation of a RuleRunner system into an equivalent logic program (Algorithm 3).

Algorithm 3 From RuleRunner to Logic Programs

```

1: function RR2LP( $\phi$ )
2:   Create  $RR_\phi = \langle R_E, R_R, S \rangle$  encoding  $\phi$ 
3:   Create an empty logic program  $LP$ 
4:   for all  $R[\phi], [\psi^1]V, \dots, [\psi^n]V, \rightarrow [\phi]V \in R_E$  do  $\triangleright C_E$ 
5:      $LP \leftarrow LP \cup [\phi]V \text{ :- } \sim[UPDATE], R[\phi], [\psi^1]V, \dots, [\psi^n]V$ 
6:   end for
7:   for all  $o \in R_E \cup R_R$  do  $\triangleright C_P$ 
8:      $LP \leftarrow LP \cup o \text{ :- } \sim[U], o$ 
9:   end for
10:  for all  $R[\phi] \in R_E \cup R_R$  do
11:     $LP \leftarrow LP \cup R[\phi] \text{ :- } \sim[UPDATE], R[\phi]$ 
12:  end for  $\triangleright C_R$ 
13:  for all  $[\phi]? \rightarrow R[\phi], R[\psi^1], \dots, R[\psi^n] \in R_R$  do
14:     $LP \leftarrow xLP \cup R[\phi] \text{ :- } [UPDATE], [\phi]?$ 
15:    for all  $R[\psi^i]$  do
16:       $LP \leftarrow LP \cup R[\psi^i] \text{ :- } [UPDATE], [\phi]?$ 
17:    end for
18:  end for
19:   $LP_\phi = LP$ 
20: return  $LP_\phi$ 
21: end function

```

The algorithm creates a single logic program LP ; however, for the sake of explanation, we distinguish three kinds of clauses: evaluation, reactivation and persistence (marked as C_E, C_R, C_P in Algorithm 3). Intuitively, evaluation and reactivation clauses (in LP) mirror, respectively, evaluation and reactivation rules in RuleRunner. Persistence clauses are used to *remember* observations and active rules by explicit re-generation: these clauses follow the pattern $x \text{ :- } \sim[UPDATE], x$, where x is a rule name ($R[\phi]$) or an observation ($o \in O$). Evaluation clauses are obtained from evaluation rules by adding one extra literal in the body, $\sim[UPDATE]$. The reactivation rules are split into several reactivation clauses, one for each literal in the head of the rule; $[UPDATE]$ is added in the body of all these rules. Finally, for all observations and truth evaluations in the rules, a persistence clause is added.

RuleRunner's monitor loop fires the evaluation and reactivation rules in an alternate fashion. We simulate that by introducing the $[UPDATE]$ literal and using it as a switch: when $[UPDATE]$ holds, only reactivation clauses can hold, while when it does not all reactivation rules are inhibited and evaluation and persistence clauses are potentially active. RuleRunner iteratively builds a state, going through partial solutions; in the same way, some sort of memory is necessary for the partial results to be remembered by the LP. We achieve that by adding persistence clauses: as long as $[UPDATE]$ does not hold, all observations and rule names are re-obtained at each iteration of the logic program. Another way to achieve this persistence is to add the desired observations/rule names as facts: we opted for the former because persistence clauses have a standard structure, while adding facts to the LP correspond to clamping neurons in a neural network.

The example in the previous section corresponds to the logic program $LP_{(a \vee \diamond b)}$, depicted below ($[UPDATE]$ shortened to $[U]$).

EVALUATION CLAUSES

- $[a]F :- \sim[U], R[a], \sim a$
- $[b]T :- \sim[U], R[b], b$
- $[b]F :- \sim[U], R[b], \sim b$
- $[\Diamond b]T :- \sim[U], R[\Diamond b], [b]T$
- $[\Diamond b]? :- \sim[U], R[\Diamond b], [b]F$
- $[a \vee \Diamond b]?_R :- \sim[U], R[a \vee \Diamond b]B, [a]F, [\Diamond b]?$
- $[a \vee \Diamond b]T :- \sim[U], R[a \vee \Diamond b]R, [\Diamond b]T$
- $[a \vee \Diamond b]?_R :- \sim[U], R[a \vee \Diamond b]R, [\Diamond b]?$
- $SUCCESS :- \sim[U], [a \vee \Diamond b]T$

REACTIVATION CLAUSES

- $R[\Diamond b] :- [U], [\Diamond b]?$
- $R[b] :- [U], [\Diamond b]?$
- $R[a \vee \Diamond b]R :- [U], [a \vee \Diamond b]?_R$

PERSISTENCE CLAUSES

- $a :- \sim[U], a$
- $b :- \sim[U], b$
- $R[a] :- \sim[U], R[a]$
- $R[b] :- \sim[U], R[b]$
- $R[\Diamond b] :- \sim[U], R[\Diamond b]$
- $R[a \vee \Diamond b]B :- \sim[U], R[a \vee \Diamond b]B$
- $R[a \vee \Diamond b]R :- \sim[U], R[a \vee \Diamond b]R$

Summarising, we start from a formal property ϕ expressed as an FLTL formula, we compute a RuleRunner rule system RR_ϕ monitoring that formula, and then we encode the rule set in a logic program LP_ϕ . The correctness of this translation can be proved by induction on the monitoring process; we omit such proof due to lack of space. We can then exploit the *CILP* algorithm [8] to translate the logic program LP_ϕ into an equivalent neural network NN_ϕ .

Algorithm 4 CILP System - Translation

```

1: function  $CILP^2P(LP_\phi)$ 
2:   Create empty network  $NN$ 
3:   for all clause  $c \in LP_\phi, c = h :- b_1, \dots, b_n$  do
4:     Add a neuron  $C$  to the hidden layer
5:     If not there, add a neuron  $H$  to the output layer
6:     Connect  $C$  to  $H$ 
7:     for all clause  $b_i \in c$  do
8:       If not there, add a neuron  $B_i$  to the input layer
9:       Connect  $B_i$  to  $C$ 
10:    end for
11:  end for
12:   $NN_\phi = NN$ 
13: return  $NN_\phi$ 
14: end function

```

Each clause (c) of LP_ϕ is mapped from the input layer to the output layer of NN_ϕ through one neuron (labelled C) in the single hidden layer of NN_ϕ . We omitted the computation of the parameters; intuitively, the Translation Algorithm from LP_ϕ to NN_ϕ has to implement the following conditions: (C1) the input potential of a hidden neuron (C) can only exceed C 's threshold, activating C , when all the positive antecedents of c are assigned the truth value true while all the negative antecedents of c are assigned false; and (C2) the input potential of an output neuron H can only exceed H 's threshold, activating H , when at least one hidden neuron C that is connected to H is activated; the translation algorithm is sound and complete [8].

$LP_{(a \vee \Diamond b)}$ is translated into the neural network $NN_{(a \vee \Diamond b)}$ in Fig. 4.

The result of the neural encoding phase is a neural network NN_ϕ able to perform runtime verification, monitoring ϕ over the given trace(s). The general algorithm for

Algorithm 5 Runtime Verification using NN_ϕ

118

5 Experiments

5.1 Parallel Monitoring

The encoding of RuleRunner in a neural network allows for the monitoring of several traces in parallel. A cell in a trace is a vector of observations which is fed to the neural network. To monitor multiple traces, it is sufficient to compose all observation vectors (of different traces at the same time) as rows in a matrix. The monitoring is then carried out as explained above, with all vector-matrix steps substituted by matrix-matrix operations. Furthermore, the matrix-based nature of neural networks allows for straightforward optimisations in order to speed up the monitoring: we focused on sparse form and GPU computation, using Matlab 2013a for all experiments. A *sparse* matrix is a matrix primarily populated by zeros (e.g. any identity matrix). The *sparse(m)* function is a lossless compression that stores only the non-null elements in a matrix m . The weight matrices in our networks are sparse: the size depends on the complexity of the encoded formula (and therefore the number of rules in the monitor), but the number of non-null elements per row/column is constant, as each rule (clause) only involves truth values of direct subformulae and has a constant number of literals in the body/head. We also exploited MathWork’s Parallel Computing Toolbox to run our code on GPU. We used GPUArray, a special array type with several associated functions that allows to perform computations on CUDA-enabled NVIDIA GPUs directly from MATLAB. In the following set of experiments, the *base* implementation for ϕ is the neural-network encoding a monitor for ϕ as described in the previous section (NN_ϕ); the *sparse* and *GPU* implementations were simply obtained, respectively, the *sparse(·)* and *gpuArray(·)* functions to all the matrices in the *base* implementation.

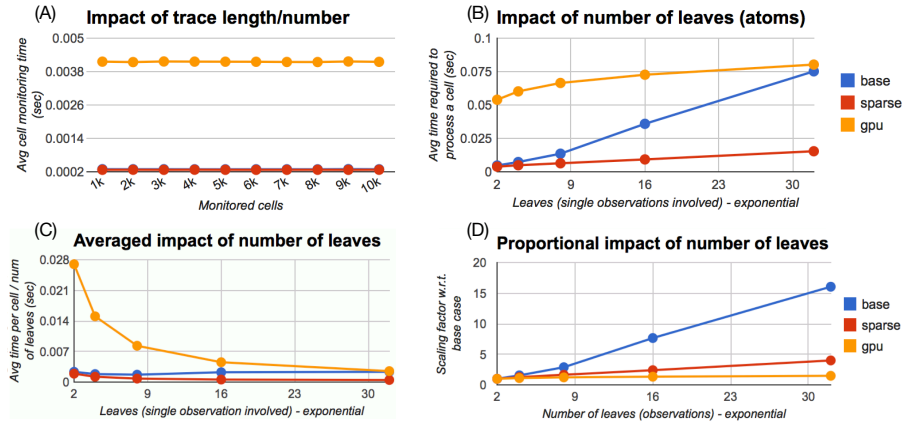


Fig. 5: Scalability experiments for matrix-based monitors

In terms of code, the difference between the three implementations is deliberately minimal, as our goal is to show how tiny modifications to the code allow for performance improvement. For this experiments, we generated random formulae from the FLTL grammar and sets of observations: rather than testing domain-specific properties, we

aimed at checking a set of unbiased formulae. The complexity of the temporal formulae is measured in terms of *leaves* of the parsing tree: for instance, $\Box(a \vee b)$ has two leaves (a, b) , $\Diamond(Xa \wedge ((c \vee d)Ud))$ has four leaves (a, b, c, d) , et cetera. In Figure 5.A, we show how the length of the trace has no impact on the average time required to monitor a cell: the resources required by our system do not grow over time. In Figure 5.B, we measured how increasing the complexity of the formula impacts on the average time required to monitor a cell. The first striking feature of these curves is that, while for simple formulae *base* and *sparse* are significantly faster than *gpu*, for higher number of leaves *base* ends up having average times similar to those of *gpu*. Moreover, *base*'s curve looks steeper than the other two. The same values can be averaged over the number of leaves: that is, given the average time required to monitor a cell when monitoring a formula with n leaves, we can divide that time value by n . The chart in Figure 5.C shows how the *gpu* curve significantly lowers the time per cell per leaf, when the number of leaves increases. Our explanation for this behaviour is that the GPU implementation suffers from a remarkable overhead in the CPU-GPU communication: this delta becomes less influent when making the formula more complex and thus the matrices bigger. Finally, it is relevant to measure the increase rate of these curves: to do so, we divided all measured times by the first experiment in each curve (Figure 5.D). We observed that *base* scales linearly w.r.t. the number of leaves: for instance, the experiment with 32 leaves is 16 times slower than the experiment (with the same implementation) with 2 leaves. The curve for the *sparse* form is more gradual, with the last experiment being ~ 4 times slower than the first one. But the implementation with the smoothest curve is the *gpu*-based one, with the 32-leaves experiment being only ~ 1.5 times slower than the 2-leaves one.

We ran other tests to study the impact of other features, such as the number of consecutive *next* operators or the interleaving of temporal and classical operators. All tests generated curves following the same patterns. We thus observed how the *sparse* form allows for a speed-up of the monitoring, while the *gpu* implementation shows the best scaling factor.

5.2 Learning as property adaptation

Neural networks also offer intrinsic learning capabilities. In our system, observations (events) are presented to the monitor as the activation of labelled input neurons. The feedforward propagation then causes some output neurons to be activated, representing the result of the monitoring of the encoded formula on the given cell. It may be the case that a user or domain expert would like to modify the verdict of a monitoring task: for instance, consider a security system built to detect consecutive *login* operations from the same user with no *logout* in between: $\phi = \text{login} \Rightarrow X(!\text{login} \cup \text{logout})$. Now suppose the system detects a number of violations, in a given LAN, that the security manager judges as false positives. For instance, that LAN could be a lab where users log in from different devices, or it could be a trustable section of the network. It is reasonable that the security manager wants to maintain the security system, but also to relax the formal property in order to include logins from the given LAN, and therefore avoid false positives and improve efficiency. Reclassifying a trace means demoting the causality relation between the observations and the actual output while promoting the relation between observations and expected output: it is, in fact, a supervised learning task.

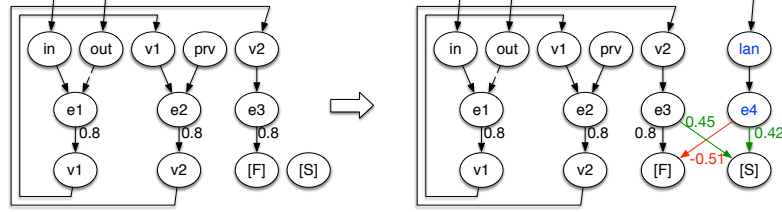


Fig. 6: Simplified network for the *consecutive logins* example, representing the current login (*in*), logout (*out*), the existence of a previous login (*prv*), the occurrence of a login without a logout before (*v1*, corresponding to a violation of $\neg \text{login} \cup \text{logout}$), the occurrence of *v1* when a previous login occurred (*v2*, corresponding to $\neg \phi$) and, finally, flags for global violation ($[F]$ for *failure*) and success ($[S]$).

The security manager tells the system to accept the consecutive logins occurring in LAN. A new input neuron, *lan* is added to the system. The learning task is performed as training of two perceptrons: $P1$ maps $\{e1, e2, e3, e4\}$ onto $[F]$, while $P2$ maps the same hidden neurons onto $[S]$. The perceptrons are conceptually isolated by the rest of the network as follows: the output is the flag to be modified, and the inputs are all hidden neurons in the network. Note that, even if all hidden neurons are used as inputs for a perceptron and its learning algorithm, only the weights of the connections from active neurons can be modified, thus limiting the algorithm to the relevant (currently active) hidden neurons. $P1$ is trained to accept only traces where $e3, e4$ occurs, while for $P2$ the positive examples involve $e3, e4$. Both perceptrons are trained with the standard perceptron learning rule: Figure 6 shows the results when applying the learning rule with $\eta = 0.2$ and thresholds = 0.5. Relevant connections are labeled with their weights; connections with weight close to 0 are omitted. For $P1$, the weights from $e1, e2, e3$ are only slightly modified by the learning; the connection from $e4$, on the contrary, goes from being 0 to a negative value. For $P2$, since the network is trained to associate the co-occurrence of the violation $v2 = \neg \phi$ and *lan*, the weights from $e3$ and $e4$ towards $[S]$ are increased. As a result, when *lan* is not active the network displays the nominal behaviour and signals a violation ($[F]$); on the contrary, when both *lan* and $v2$ are active, the negative weight from $e4$ prevents $[F]$ from being activated, while the sum of the signals from $e3$ and $e4$ activates $[S]$.

6 Conclusions and Future Work

Firstly, we developed a runtime monitoring system, RuleRunner, to monitor FLTL properties on finite traces. Secondly, we discussed the encoding, partially based on CILP's Translation Algorithm, of a RuleRunner rule system in a standard feedforward neural network with recurrent connections and one hidden layer. Thirdly, we showed how matrix-based optimisations allows for major improvements in terms of performance, either concerning actual speed (matrices in sparse form) or scaling factor (GPU computation). Finally, we pointed out how in our framework learning corresponds to property adaptation. A first direction for future work is to adopt and compare more advanced techniques for

parallel monitoring, such as the MapReduce framework (recently applied to Runtime Verification in [1]) and the parallel algorithms in [5]. A second direction for future work is the development of ad-hoc learning strategies, mostly based on the Backpropagation algorithm [11], for specific classes of adaptation problems, such as soft norms in BPM [19] and false positives in Security [6].

References

1. B. Barre, M. Klein, M. Soucy-Boivin, P.-A. Ollivier, and S. Hall. Mapreduce for parallel trace validation of ltl properties. In *Procs. of Runtime Verification 2012*, pages 184–198.
2. H. Barringer, D. E. Rydeheard, and K. Havelund. Rule systems for run-time monitoring: from eagle to ruler. *Journal of Logic and Computation*, volume 20:pages 675–706, 2010.
3. A. Bauer, M. Leucker, and C. Schallhart. The good, the bad, and the ugly, but how ugly is ugly? In *Procs. of Runtime Verification 2007*, pages 126–138.
4. A. Bauer, M. Leucker, and C. Schallhart. Monitoring of real-time properties. In *Procs. of Foundations of Software Technology and Theoretical Computer Science 2006*, pages 260–272.
5. S. Berkovich, B. Bonakdarpour, and S. Fischmeister. Gpu-based runtime verification. In *Procs. of the IEEE International Symposium on Parallel and Distributed Processing 2013*, pages 1025–1036.
6. D. Breitgand, M. Goldstein, and E. H. Shehory. Efficient control of false negative and false positive errors with separate adaptive thresholds. *Network and Service Management, IEEE Transactions on*, 8:128–140, 2011.
7. A. Coates, B. Huval, T. Wang, D. J. Wu, B. C. Catanzaro, and A. Y. Ng. Deep learning with cots hpc systems. In *Procs. of International Conference on Machine Learning 2013*, pages 1337–1345.
8. A. S. d’Avila Garcez and G. Zaverucha. The connectionist inductive learning and logic programming system. *Applied Intelligence*, volume 11:pages 59–77, 1999.
9. D. Drusinsky. The temporal rover and the atg rover. In *Procs. of the International Workshop on SPIN Model Checking and Software Verification 2000*, pages 323–330.
10. C. Eisner, D. Fisman, J. Havlicek, Y. Lustig, A. McIsaac, and D. V. Campenhout. Reasoning with temporal logic on truncated paths. In *Procs. of Computer-Aided Verification 2003*, pages 27–39.
11. S. Haykin. *Neural Networks: A Comprehensive Foundation (3rd Edition)*. 2007.
12. P. Heimann, G. Joeris, C.-A. Krapp, and B. Westfechtel. Dynamite: Dynamic task nets for software process management. In *Procs of International Conference on Software Engineering 1996*, pages 331–341.
13. Q. V. Le, M. Ranzato, R. Monga, M. Devin, G. Corrado, K. Chen, J. Dean, and A. Y. Ng. Building high-level features using large scale unsupervised learning. In *Procs. of International Conference on Machine Learning 2012*.
14. M. Leucker and C. Schallhart. A brief account of runtime verification. *Journal of Logic and Algebraic Programming*, volume 78:pages 293–303, 2009.
15. O. Lichtenstein, A. Pnueli, and L. D. Zuck. The glory of the past. In *in Procs. of Logic of Programs 1985*, pages 196–218.
16. J. Lukasiewicz. *O logice trjwartosciowej (On Three-Valued Logic)*. 1920.
17. A. Pnueli. The temporal logic of programs. In *Procs. of the Annual Symposium on Foundations of Computer Science 1977*, pages 46–57.
18. G. G. Towell and J. W. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, volume 70:pages 119–165, 1994.
19. W. M. P. van der Aalst et.al. Process mining manifesto. In *Procs of Business Process Management Workshops 2011*, pages 169–194.

Using Monotonicity to find Optimal Process Configurations Faster

D.M.M. Schunselaar^{1*}, H.M.W. Verbeek^{1*}, H.A. Reijers^{1,2*}, and W.M.P. van der Aalst^{1*}

¹ Eindhoven University of Technology,
P.O. Box 513, 5600 MB, Eindhoven, The Netherlands
{d.m.m.schunselaar, h.m.w.verbeek, w.m.p.v.d.aalst, h.a.reijers}@tue.nl
² VU University Amsterdam,
De Boelelaan 1105, 1081 HV Amsterdam, The Netherlands
h.a.reijers@vu.nl

Abstract. Configurable process models can be used to encode a multitude of (different) process models. After configuration, they can be used to support the execution of a particular process. A configurable process model represents a space of instantiations (configured process variants). Such an instantiation space can be used by an organisation to select the best instantiation(s) according to some Key Performance Indicators (KPIs), e.g., cost, throughput time, etc. Computing KPIs for all the instantiations in the space is time consuming, as it might require the analysis (e.g., simulation) of thousands (or more) of instantiations. Therefore, we would like to exploit structural characteristics to reduce the amount of instantiations which need to be analysed. This reduction only removes those instantiations which do not need to be considered by an organisation. This yields the same result (a collection of best configurations), but in a faster way.

Keywords: Configurable Process Model, Business Process Performance, Analysis, Monotonicity, Petra

1 Introduction

To go abroad, travellers typically need to have a passport. To obtain a new passport, a traveller needs to go to his own municipality. For example, Fry has to go to the New New York municipality, while Homer has to go to the Springfield municipality. Although the New New York municipality will create a passport for Fry, they will not do so for Homer, as he is not living in New New York, but in Springfield. As such, the municipalities all offer the service to create a

* This research has been carried out as part of the Configurable Services for Local Governments (CoSeLoG) project (<http://www.win.tue.nl/coselog/>).

new passport, but they are not competing with each other as they only offer this service for their own inhabitants. As a result of the latter, municipalities are quite eager to share their processes with other municipalities, and to learn from each other. Although all municipalities offer the service to create a new passport, they do not all use the exact same process. Some “couleur locale” may exist between different municipalities. For example, the New New York municipality may create the passport first and have Fry pay when he collects his passport, while the Springfield municipality, being smaller, may require Homer to pay when he requests for it, that is, before they create it. Even though the steps in the process may be the same (request for a passport, pay for it, create it, and collect it), the process may still differ to some extent.

The combination of this “couleur locale” and the openness mentioned earlier makes municipalities natural candidates to benefit from co-called configurable process models. A configurable process model contains variation points which can be set to tailor the configurable process model to the preferences of an organisation. Setting preferences for the variation points is called a configuration. If all variation points are set by a configuration, the resulting process model is called an instantiation. Please note that a configurable process model that contains a multitude of variation points allows for an exponential amount of possible configurations and instantiations.

A configurable process model may contain instantiations that are not used by any of the municipalities. An interesting question now is, whether some instantiations score better for a given municipality than the instantiation they are now using. Given a set of Key Performance Indicators (KPIs), we could check how every instantiation scores on these KPIs, and return the corresponding best configurations. In [1], we have introduced *Petra*, a generic framework that automatically analyses KPIs on all instantiations of a configurable process model for instance by simulating all the instantiations. By exhaustively searching *all* instantiations, *Petra* returns a Pareto front [2] of the best instantiations to the municipality, which can then select the configuration they like best.

As mentioned earlier, a configurable process model may allow for many variation points and, as a result, very many configurations and instantiations. As a result, the amount of instantiations may be too large to analyse, or it may simply take too much time to analyse them all. Therefore, in this paper, we aim to reduce the amount of instantiations which need to be analysed by exploiting the fact that they all stem from the same configurable process model. For example, if we take the passport example as introduced earlier, it is clear that the Springfield instantiation allows for less financial risks than the New New York one. For this reason, if financial risk would be the KPI at hand, it would make no sense to analyse the New New York instantiation, as it will be dominated anyways by the Springfield instantiation.

To achieve this reduction in the amount of instantiations to analyse, we propose to exploit structural properties of the configurable process model by means of a *monotonicity* notion. This paper introduces this monotonicity framework, applies it for a concrete KPI, and evaluates the application empirically on an

artificial configurable process model. The monotonicity framework creates, per KPI, an ordering of the instantiations. This ordering starts with the instantiation most probably to have a high score on that KPI. Using these orderings, the monotonicity framework starts analysing the most promising instantiations. It keeps on analysing until no more instantiations can be found which score higher on a KPI than the already analysed models.

For our concrete KPI, we have selected throughput time and we show how the monotonicity can be computed between two instantiations. We apply this monotonicity notion on a running example to order the instantiations. Afterwards, we use simulation to obtain concrete values for the KPI. Using these values, we can conclude that we achieve a reduction of more than 90% on the amount of instantiations which need to be analysed for this particular model.

The remainder of this paper is organised as follows. In Sect. 2, we elaborate on related work. Afterwards, we present some preliminaries in Sect. 3. Sections 4 and 5 contain our monotonicity framework and concrete results for the concrete throughput time KPI. We finish the paper with an empirical evaluation and the conclusions and future work in Sect. 6 and Sect. 7.

2 Related Work

Our research builds on three existing lines of research: configurable process models, performance analysis, and business process reengineering.

2.1 Configurable Process Models

Configurable process models (see Sect. 3.1 for an example configurable process model) have been developed by extending existing modelling languages, e.g., C-EPC (Configurable Event-driven Process Chain) [3], C-BPEL (Configurable Business Process Execution Language), and C-YAWL (Configurable Yet Another Workflow Language) [4]. A more complete overview of variability support is provided in [5]. All these approaches are mainly focussed on supporting variability and not so much on the analysis of the resulting instantiations.

2.2 Performance Analysis

Within the field of queueing theory, work has been conducted in defining monotonicity notions between queueing networks and between queueing stations. In [6], the author defines a notion between queueing stations and between queueing networks for closed queueing networks. The definition of monotonicity employed is similar to our notion of monotonicity. However, this paper is mainly focussed on the parameters of the network (number of jobs, processing speed of networks) and not on the relation between two topologically different networks.

The authors in [7] consider performance monotonicity on continuous Petri nets. Similar to the work in [6], the authors consider monotonicity in terms of the parameters of the Petri net and not in terms of the structure of the Petri

net. However, since our used formalism can be translated to Petri nets, this is an interesting approach to consider for future work.

Although the papers consider monotonicity in a similar way as us, they focus on the parameters instead of the topology of the network. However, one of our future directions is to take the parameters also into account in which this work might be applicable.

2.3 Business Process Reengineering

In [8], the author presents a tool KOPeR (Knowledgebased Organizational Process Redesign) for identifying redesign possibilities. These redesign possibilities are simulated to obtain performance characteristics such that they can be compared. The analysis of process models focusses mainly on the analysis of a single model and not on various instantiations. An approach evaluating when certain changes to the structure of the process model are appropriate is presented in [9]. The paper starts from a number of commonalities in reengineered business processes and deduces, based on queueing theory models, under which circumstances a change to the structure of the process model is beneficial. Some ideas of their paper can be applied to our setting but the majority of ideas is not tailored towards throughput time.

In [10], so-called Knock-Out systems are discussed and heuristics are defined for optimising these. Similar to [9], heuristics are defined and formally shown if it is beneficial to apply a certain heuristic in a particular setting. Their approach allows for more flexibility in defining the processes than configurable process models, i.e., in the paper only a precedence relation is defined between tasks. At the same time, the processes considered are less flexible as they do not include choices, i.e., every task has to be executed. As with the previous approach, some ideas can be used in our approach.

The approach closest to our approach is presented in [11]. In their approach, various process alternatives are analysed. These alternatives are obtained by applying redesign principles instead of starting from a configurable process model. Their approach can benefit of the work presented here as it might remove the need to analyse some instantiations due to monotonicity.

3 Preliminaries

Before introducing the monotonicity framework, this section introduces the configurable process models used by the framework and the *Petra* framework.

3.1 Configurable Process Models

Configurable process models contain predefined variation points. By configuring these variation points, that is, by selecting appropriate values for these points, the configurable process model can be tailored to an organisation (like a municipality). If all variation points have been configured properly, that is, if the

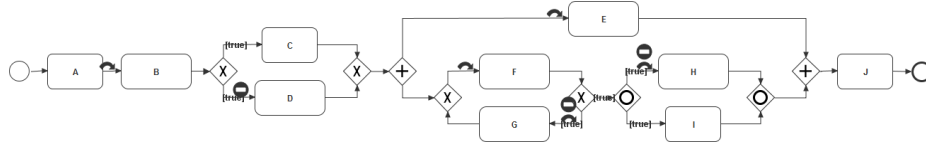


Fig. 1: Running example of a configurable process model in BPMN notation.

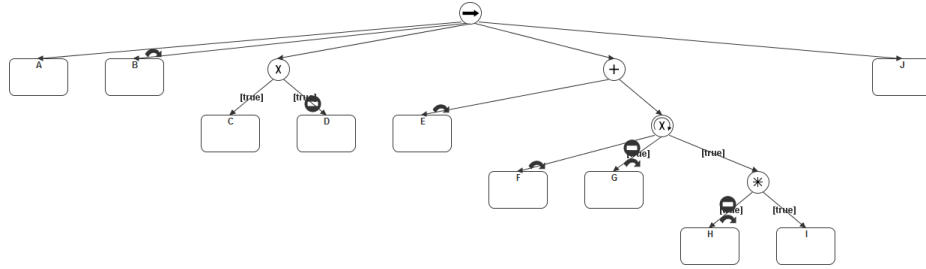


Fig. 2: Running example from Fig. 1 in Process Tree notation.

configuration is complete, the configurable process model is instantiated into a process model ready that is ready for enactment (an instantiation).

As an example, Fig. 1 shows the control-flow of a configurable process model using a BPMN (Business Process Model and Notation) notation augmented with curved arrows and no-entry signs. A curved arrow on an incoming edge indicates a variation point that allows the following part of the configurable process model to be hidden. For example, the curved arrow on the incoming edge of the task labelled “B” indicates that this activity can be hidden. Note that if this curved arrow would have been positioned on the outgoing edge of this task, that then the entire following choice part, including the tasks labelled “C” and “D”, would then have the option to be hidden. Likewise, the no-entry sign indicates a variation point that allows the following part of the configurable process model to be blocked.

If a part of the configurable process model is hidden, this results in this part being substituted by an automatic task. If a part of the configurable process model is blocked, this results in removing this part in total. As a result, if a part of a sequential execution of tasks is blocked, then the entire sequence is blocked, as we would run into a deadlock otherwise.

For our configurable process model formalism, we use so-called Process Trees [1]. Figure 2 shows the Process Tree representation of the process model as shown in Fig. 1. A Process Tree is a block-structured process modelling formalism and is specifically developed in the CoSeLoG project. The main advantage of Process Trees over other formalisms is that it ensures soundness, e.g., there cannot be any deadlocks [12]. The various node types in the Process Tree and their semantics in BPMN are depicted in Fig. 3. Nodes come in three flavours; tasks, blocks, and events. Tasks form the units of work and can be either automatic (without

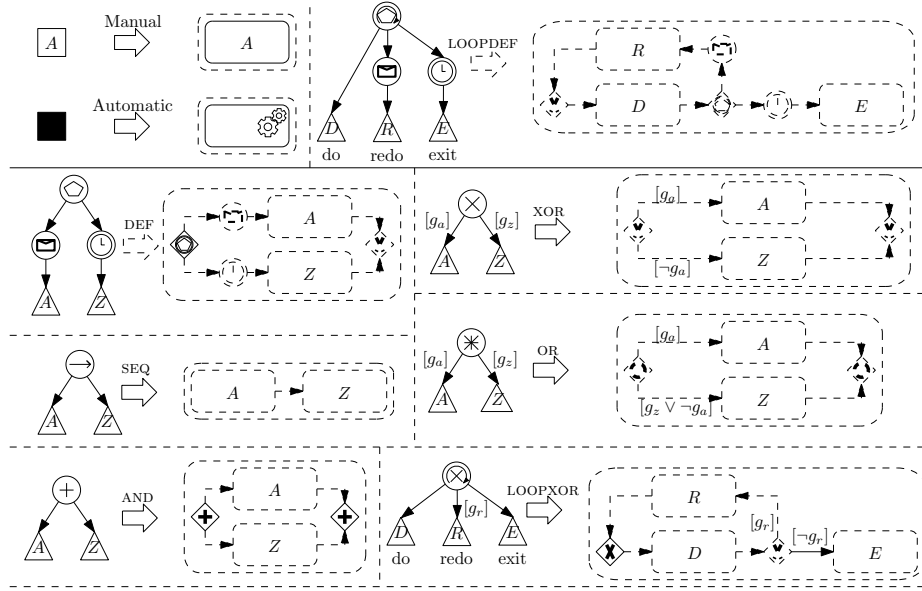


Fig. 3: The various nodes of a Process Tree and their BPMN semantics. For choices we have a default option being the last child. This means that if all expressions evaluate to false the last branch is chosen. For the XOR, this is encoded by annotating the last branch with the negation of the other branch. Note that we have shown the case with two children. It is trivial to extend this to more children.

a resource) or manual (with a resource). Blocks indicate the causal dependency of the children, i.e., the nodes directly underneath the block. Events indicate a point in the process where input from the environment is required. Only events that match the actual input will be executed, the other events will be dropped. In principle a block can have any number of children except for the LOOP nodes (LOOPXOR and LOOPDEF), which always have 3 children, and the EVENT nodes, which have a single child. At the top of the Process Tree we have a root node.

Next to hiding and blocking, a Process Tree allows for a third type of variation points, called placeholder nodes. Where hiding and blocking can be configured by selecting either “yes” or “no”, a placeholder node can be configured by selecting one of its child nodes. As a result, the placeholder node will be replaced in an instantiation by its selected child node. For instance, in a configurable process model, there may be the possibility to select a payment method from a set of known payment methods (credit card, bank transfer, or cash). The Process Tree formalism is richer than just the control-flow perspective [1], but here we limit ourselves to the control-flow perspective, i.e., we assume the other perspectives remain unchanged.

3.2 Petra

Petra [13] (Process model based Extensible Toolset for Redesign and Analysis) is a framework for analysing configurable process models. *Petra* employs an iterative brute force approach in traversing the instantiations of a configurable process model. In each iteration, *Petra* chooses an instantiation and applies various analysis tools to this. As a result, the values for the required KPIs become known for this instantiation, and this instantiation can be added at the specific point in the Pareto front [2]. As the Pareto front only keeps track of the best (non-dominated) points, instantiations that may have been added may be removed at some point in time. In the end, only those instantiations that are not dominated by some other instantiation will survive on the Pareto front. The sets of tools and KPIs within *Petra* are extensible with new tools and KPIs. As a result, we prefer not to limit the monotonicity notion to a predetermined set of KPIs.

4 Monotonicity Framework

The monotonicity framework stems from the observation that it may be possible to check whether an instantiation dominates another instantiation by comparing the structure, control flows in or case, instead of the behaviour. This means that we want to compare two models with respect to a KPI without computing the actual values for that KPI. Consider, for example, two instantiations from the running example (see Fig 2), and assume that for the first instantiation nothing has been blocked or hidden, and that for the second nothing has been blocked and only the task labelled “B” has been hidden. Clearly, the throughput time of the second instantiation is always better than the throughput time of the first, as the only difference is that the first has to execute “B”, which we assume takes some time, where the second does not. Based on structure, we could claim that the second instantiation is better as the first w.r.t. the throughput KPI under all circumstances. As a result, when looking only at the throughput KPI, the first cannot be better than the latter (assuming independence between the duration of an activity and the occurrence of another).

For this reason, the monotonicity framework introduces an acyclic (partial) order on the possible instantiations. Throughout the paper, we use the term “at-least-as-good” to denote the monotonicity ordering between instantiations/nodes for a particular KPI. We define this relation formally as:

Definition 1 (At-least-as-good). *A node n is at-least-as-good as another node n' (denoted $n \geq n'$) w.r.t. KPI K if $\forall_c P[K(n) \leq c] \leq P[K(n') \leq c]$, i.e., the cumulative distribution function (CDF) of the distribution $K(n)$ is for every point c at most the CDF of the distribution of $K(n')$ in that point. An instantiation M is at-least-as-good as another instantiation M' w.r.t. KPI K if and only if the root node of M is at-least-as-good as the root node of M' w.r.t. K .*

Note that it is possible that individual values of $K(n')$ are better than $K(n)$, but overall the values for n are better than n' .

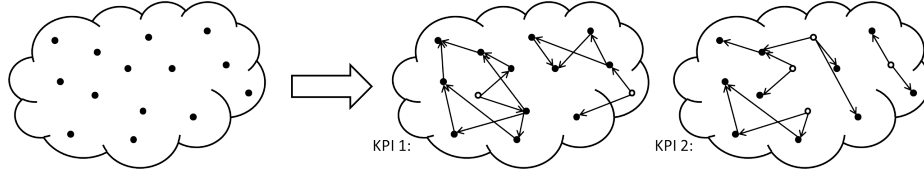


Fig. 4: Using monotonicity, we can transform the set of possible instantiations to a partial order indicating the most promising order of analysing the instantiations for various KPIs.

If we have that a model M is at-least-as-good as a model M' for all (relevant) KPIs, then clearly M should dominate (or at least equal) M' and as a result it only makes sense to analyse M' if M needs to be analysed. Later on, we will see how we can derive this at-least-as-good relation. First, however, we will show how *Petra* uses this relation.

Graphically, the monotonicity transforms the collection of possible instantiations to a collection of possible related instantiations. Since there might be multiple KPIs in the framework, we obtain (different) relations for each of the KPIs (Fig. 4). The dots are the instantiations and an arrow between two dots indicates that the instantiation at the tail of the arrow is at-least-as-good as the instantiation at its head. The open dots indicate the most promising instantiations. By transitivity, if there exists a directed path from one instantiation to another instantiation, then the former instantiation is at-least-as-good as the latter w.r.t. the corresponding KPI.

With the monotonicity framework added, *Petra* analyses the possible instantiations in a specific order. If M is at-least-as-good as M' on all KPIs, then M will be analysed by *Petra* before M' will be analysed. If M' is to be analysed by *Petra* and if at that point in time M has been dominated by some other model M'' , then M' cannot dominate M'' and there is no use in analysing it. Otherwise, if M is not dominated, M' is analysed by *Petra*.

When an instantiation is dominated by other models, it creates a cut-off point along the partial orders for the various KPIs, as this instantiation is at-least-as-good (w.r.t. the KPI at hand) as every instantiation that can be reached by a directed path. As a result, if an instantiation is below the cut-off points for all KPIs, it is dominated by the previously analysed models and there is no use in analysing it.

To determine whether an instantiation is at-least-as-good as another instantiation, we need to check whether its root node is at-least-as-good-as the other root node. To determine this, we use a bottom-up approach, which uses the fact that both are instantiations of the same configurable process model. As a result of this, we can relate two nodes in both instantiations in a straightforward way by determining whether they stem from the same node in the configurable process model: They are related if and only if they stem from the same node. Note that hiding a part of the configurable process model results in an automatic task

with duration 0 in the instantiation. As a result, such an automatic task can be related to any other node. Furthermore, note that if a placeholder node is configured as a node of one type (like SEQ) in one instantiation, and as a node of another type (like AND) in another instantiation, it is possible that a node of one type is related to another node of another type.

For a task node, it is usually quite straightforward to check whether or not they are at-least-as-good as their related nodes: As both stem from the same node, they are equal, and hence at-least-as-good. For a block node, we need to look whether all relevant child nodes are at-least-as-good as their related nodes, which is where the bottom-up approach comes in. Based on the structures of both instantiations and using the fact which child nodes are at-least-as-good as their related nodes, we determine whether a give node in one instantiation is at-least-as-good as its related node in the other instantiation. With this approach, we decompose the problem into smaller problems and basically use patterns to identify which of the elements is better. If we can conclude that the one root node is at-least-as-good as the second root node, then we can conclude that the one process model is at-least-as-good as the second process model.

In our monotonicity framework, we assume there is a correspondence between node types and the effects on the value of a KPI. This stems from the observation that some KPIs behave monotone in two flavours, i.e., more is better (monitoring activities in the process for compliance) or less is better (costs). KPIs which do not behave monotone in that respect, e.g., wait time which can increase and decrease with the addition/removal of activities, cannot be captured in this framework.

Because we only take the structure of the instantiations into account, we restrict ourselves to the control-flow perspective in this paper. However, to be able to compare choice nodes (XOR, DEF, OR), we assume there is a probability associated with the outgoing edges of the choice node. This probability indicates the (relative) probability that the flow of control follows that path. Likewise, to be able to compare LOOP nodes (LOOPXOR and LOOPDEF), we assume that there is a probability associated with the outgoing edges to the REDO and EXIT blocks. In the next section, we demonstrate the applicability of our approach by focussing on a single KPI.

5 Throughput Time

The example KPI to be used in our monotonicity framework is the throughput time (sometimes also called sojourn time or lead time). The throughput time is the time it takes a case from start to end. We have chosen the throughput time since this KPI is well-studied and often considered for analysing process models. Using our monotonicity framework, we need to be able to take two instantiations (two Process Trees) and decide which instantiation is at-least-as-good (if any). As mentioned, we focus on the control-flow perspective. Therefore, we assume the other perspectives do not change between different instantiations. However, we do not disregard the other perspectives as this might lead to counter-intuitive

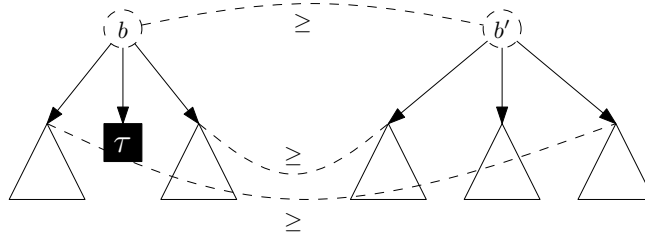


Fig. 5: The general constraints for comparing two block nodes. Note that if the child of b is a silent task (black square with white τ), it does not need to be related to a child of b' .

results, e.g., if we have a choice between a fast and a slow branch, then reducing the amount of work for the slow branch and increasing the amount of work for the fast branch might actually increase the throughput time since the fast branch cannot handle more work. Therefore, we focus on reducing the amount of work for branches without increasing the amount of work for other branches. We go through the collection of nodes and present when a node is at-least-as-good as the related node (w.r.t. throughput time).

5.1 Tasks

A silent task (an automatic task with duration 0) is always at-least-as-good as any node, and can be ignored (when not related) in a SEQ or AND block. Any other automatic task can be compared according Def. 1 with another automatic task. In all other cases, we cannot say whether an automatic task is at-least-as-good as the other node.

A manual task is at-least-as-good as the same manual task. In all other cases, we cannot say whether a manual task is at-least-as-good as the other node. Please note that, as mentioned earlier, we only take the control-flow perspective into account. If we would take the resource perspective into account, then we could check whether the same manual task would be performed by generally faster or less overloaded employees.

5.2 Blocks

In Fig. 5, the general case is depicted where every comparison between block nodes has to adhere to. A block node b is at-least-as-good as a related block node b' if every child node c of b (except for silent tasks) is related to a child node c' of b' such that node c is at-least-as-good as node c' .

The general case is sufficient for the SEQ, AND, and EVENT nodes which are related to nodes of the same type. Next to this, if b is an AND node and b' is a SEQ node and the general case holds, then we can also conclude that b is at-least-as-good as b' since doing things in parallel is at-least-as-good as doing things in sequence for the throughput time. Finally, if b is a SEQ or AND node and

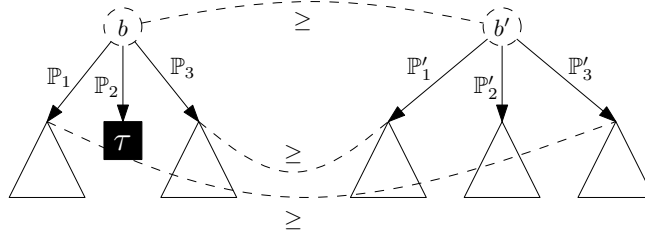


Fig. 6: The general case but now the edges are annotated with probabilities.

b' is a LOOP node (LOOPXOR, LOOPDEF) and the DO and the EXIT are the only children related of the LOOP, then b is at-least-as-good as b' . This comparison to the LOOP stems from the fact that DO is executed at least once and is eventually followed by EXIT. Thus they are in a sequence.

For choices, we need more information, i.e., we need to know the probability of executing a particular child. Therefore, we extend the general case with probabilities yielding Fig. 6. Note that implicitly the general case also contains probabilities but these are all 1, e.g., in a sequence there is no option to not execute a particular child.

Comparing two XOR/DEF nodes with each other requires that, apart from the comparison of the general case, the probabilities for the related nodes are the same. Note that in the general case, unrelated children of b are only allowed to be silent tasks, which means that these have a throughput time of 0 making them at-least-as-good as any unmapped child of b' . From this, with the requirement of equal probability between the related nodes, we know the same fraction of cases goes to unrelated nodes in both b and b' . For this fraction of cases, we know the unrelated children in b are at-least-as-good as the unrelated children in b' . Comparing two OR nodes is similar to two XOR/DEF nodes, only the probabilities of the children in b have to be at most the probabilities of the related children of b' . Note that the sum of the probabilities on the outgoing edges of an OR is at least 1. The reasoning behind this at most is that more cases having to be executed by a particular node (i.e., a higher probability) does not lower the throughput time and thus is that node at-least-as-good as the related node. This also holds when comparing a XOR/DEF with an OR, i.e., the probabilities of the children of the XOR/DEF have to be at most the probabilities of the related children in OR whilst adhering to the general case.

Next to comparing choices with each other, we can also compare choices to SEQ and AND using the earlier observation that the children of the SEQ and AND have implicitly a probability of 1. The rules are the same as for the comparison with the OR, i.e., the probabilities are at most the probabilities of the SEQ and AND, and the general case is adhered to.

We can also compare choices to LOOP nodes. For this it is sufficient to adhere to the general case and the probability of the child of b related to the REDO should be at most the probability of the REDO in b' . The intuition behind this is that the probability of the DO and EXIT are both 1, i.e., they both are executed at

Table 1: The rules the combinations of nodes have to adhere to in order to deduce that b is at-least-as-good as b' . An explanation of the used numbers can be found at the bottom.

$b \backslash b'$	SEQ	AND	EVENT	LOOP	XOR/DEF	OR
SEQ	0	-	-	1	-	-
AND	0	0	-	1	-	-
EVENT	-	-	0	-	-	-
LOOP	-	-	-	4	-	-
XOR/DEF	0	0	-	4	2	3
OR	0	0	-	4	-	3
0: general case						
1: general case and only the DO and EXIT are mapped.						
2: general case and the probabilities of mapped nodes are equal.						
3: general case and the probabilities are at most the probability of the mapped node.						
4: general case and the probability for the REDO is at most the probability of the mapped node						
-: not (yet) supported.						

least once. Thus we have to make sure the child related to the REDO is executed at most as often as the REDO, i.e., the probability of the node related to the REDO is at most the probability of the REDO.

Finally, in order to compare two LOOP nodes, we need to have the general case. On top of this, we need that the probability of executing the REDO of b is at most the probability of executing the REDO of b' . The idea behind this is that the higher the probability of the REDO, the more often the LOOP will be executed yielding a higher throughput time.

The requirements on the relation between two blocks are summarised in Table 1. The numbers indicate which requirements are to be adhered to in order for b to be at-least-as-good as b' . An explanation of the numbers is at the bottom of Table 1.

6 Empirical Evaluation

We have chosen an empirical evaluation over an asymptotic analysis since worst-case we still have to analyse all the possible instantiations. This comes from the fact that some models are incomparable (due to choices), and that, although some are at-least-as-good, the models have values for a KPI which are too close to each other making none of the models strictly better than another model.

For our empirical evaluation, we use the configurable process model from Fig. 2. We want to show that we can prune a significant part of the instantiation space prior to analysis. To analyse an instantiation, we simulate it at least 30

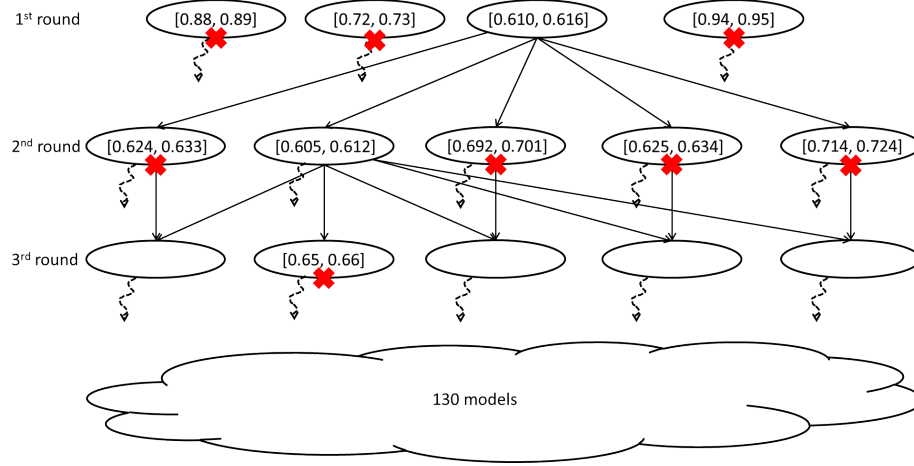


Fig. 7: The various rounds of analysis with the instantiations which were analysed and their 95% confidence intervals. We started in round 1 with the 4 most promising instantiations. In round 2, we continued with the 5 most promising instantiations from the only model that survived the first round. Finally, in round 3, we could conclude that all other instantiations are dominated.

times using L-SIM: a simulation tool developed by Lanner³. To enable simulation, we have extended our configurable process model with resource and timing information.

There are 144 possible instantiations from our running example (Fig. 2). Thus the collection of possible instantiations in Fig. 4 contains 144 dots. In Fig. 7, the various analysis rounds of our approach are depicted. Each round corresponds to analysing a group of instantiations which do not share an at-least-as-good relation and for which all instantiations that are at-least-as-good have already been analysed and are not dominated (yet) by another model. In the first round, we start with 4 instantiations (depicted by the 4 ovals at the top) which were most promising, i.e., there was no instantiation which was at-least-as-good as one of these 4 instantiations.

After simulating the most promising instantiations (the 95% confidence intervals of the throughput times are depicted in the ovals), only 1 of these instantiations was significantly better than the other instantiations and was kept as one of the best models. For the second round, we obtained 5 other models which were most promising (and not yet analysed) from our monotonicity. Simulating each of these models resulted in 1 model being better than the other models in the second round. This model was added to the set of best models. In the third round, again 5 models were most promising and not yet analysed. By our monotonicity notion, we knew 4 of them did not need to be analysed as non-best models from the second round were at-least-as-good as these 4. The remaining

³ <http://www.lanner.com/en/l-sim.cfm>

model was simulated and was significantly worse than the best models. Since none of the models from the third round made it to the set of best models, we could conclude that all other 130 instantiations were dominated. Therefore, there was no need to analyse the other models.

From the 144 instantiations, we only had to analyse 10, which means that only 6.9% of the possible instantiations had to be analysed. Analysing all models took a bit more than 50 minutes on a single core of 2.80 GHz (including some I/O handling). The average time per model is a little bit more than 20 seconds. Computing the monotonicity of the 144 models took a bit more than 2 seconds. Only analysing the 10 models, took around 3 minutes. Note that, in Def. 1, we used the CDF for determining whether one model is at-least-as-good as another model. Since with simulation we cannot determine the CDF, we have used the confidence intervals as an approximation of this CDF.

7 Conclusions and Future Work

Within *Petra*, we analyse large amounts of instantiations from a configurable process model. These analysed instantiations are projected on a Pareto front to only keep the instantiations that are most promising, according to some Key Performance Indicators (KPIs), for an organisation. Due to the possible large amount of variation point in a configurable process model, and the resulting very large amount of possible instantiations, analysing each and every instantiation is very time consuming and unnecessary as most will never be considered by an organisation.

To prevent having to analyse all possible instantiations, using the fact that most instantiations will never be considered, we sort them according to their likelihood of appearing on the Pareto front. The sorting of the instantiations happens using our monotonicity framework. This framework can be extended to work with a multitude of KPIs.

We have applied our framework with a concrete KPI (throughput time) on the configurable process model that was used as running example, and have shown that we can achieve a significant decrease in the amount of instantiations which need to be analysed (exceeding 90%). But these results are highly dependent on the model and on the characteristics of the KPIs.

This work shows promising results and we plan to extend this into a multitude of directions. We briefly sketch a panorama of future extensions. Currently, we still need to traverse the entire instantiation space to compute the ordering between models. In the ideal case, we can constructively create the configuration for the instantiations most promising for a particular KPI. Next to this, we also want to incorporate more KPIs into the framework. Furthermore, we want to generalise this work to also be able to compute monotonicity between two process models which are not necessarily instantiations from a single configurable process model. Finally, we want to leverage some of the related work which defines monotonicity on the parameters of the configurable process model to our framework.

References

1. Schunselaar, D.M.M., Verbeek, H.M.W., Aalst, W.M.P. van der, Reijers, H.A.: Petra: Process model based Extensible Toolset for Redesign and Analysis. Technical Report BPM Center Report BPM-14-01, BPMcenter.org (2014)
2. Kung, H.T., Luccio, F., Preparata, F.P.: On finding the maxima of a set of vectors. *J. ACM* **22**(4) (1975) 469–476
3. Rosemann, M., Aalst, W.M.P. van der: A Configurable Reference Modelling Language. *Information Systems* **32**(1) (2007) 1–23
4. Gottschalk, F., van der Aalst, W.M.P., Jansen-Vullers, M., Rosa, M.L.: Configurable workflow models. *International Journal on Cooperative Information Systems* **17**(2) (2008) 177–221
5. Ayora, C., Torres, V., Weber, B., Reichert, M., Pelechano, V.: Vivace: A framework for the systematic evaluation of variability support in process-aware information systems. *Information and Software Technology* (0) (2014) –
6. Suri, R.: A concept of monotonicity and its characterization for closed queueing networks. *Operations Research* **33**(3) (1985) pp. 606–624
7. Mahulea, C., Recalde, L., Silva, M.: Basic server semantics and performance monotonicity of continuous petri nets. *Discrete Event Dynamic Systems* **19**(2) (2009) 189–212
8. Nissen, M.E.: Redesigning reengineering through measurement-driven inference. *MIS Quarterly* **22**(4) (1998) 509–534
9. Buzacott, J.A.: Commonalities in reengineered business processes: Models and issues. *Manage. Sci.* **42**(5) (May 1996) 768–782
10. van der Aalst, W.M.P.: Re-engineering knock-out processes. *Decision Support Systems* **30**(4) (2001) 451–468
11. Netjes, M.: Process Improvement: The Creation and Evaluation of Process. PhD thesis, Eindhoven University of Technology (2010)
12. van der Aalst, W.M.P., van Hee, K.M., ter Hofstede, A.H.M., Sidorova, N., Verbeek, H.M.W., Voorhoeve, M., Wynn, M.T.: Soundness of workflow nets: classification, decidability, and analysis. *Formal Asp. Comput.* **23**(3) (2011) 333–363
13. Schunselaar, D.M.M., Verbeek, H.M.W., van der Aalst, W.M.P., Reijers, H.A.: Petra: A tool for analysing a process family. In Moldt, D., Rölke, H., eds.: International Workshop on Petri Nets and Software Engineering (PNSE’14). Number 1160 in CEUR Workshop Proceedings, Aachen, CEUR-WS.org (2014) 269–288 <http://ceur-ws.org/Vol-1160/>.

Tracking Hot Topics for the Monitoring of Open-world Processes

Remo Pareschi¹, Marco Rossetti², Fabio Stella²

¹ Department of Bioscience and Territory, University of Molise, Pesche (IS), Italy
remo.pareschi@unimol.it

² Department of Informatics, Systems and Communication, University of Milano-Bicocca,
Milan, Italy
{rossetti, stella}@disco.unimib.it

Abstract. We introduce the notion of open-world process to refer to processes that generally require flexible execution and are influenced by external factors. Among such processes we focus on those that fit also with the notion of “business process”. We then introduce “hot topics” to capture contextual information flows that, by flowing into the context of execution of open-world business processes, may affect their definitions. Hot topics are discovered using unsupervised learning techniques based on Probabilistic Topic Modeling and by tracking variations of the information flows into topics over time. We illustrate an application of this methodology to the tracking of recent innovations in labor laws that affect a variety of open-world business processes, from labor sourcing to merge-and-acquisition. Finally, we define a number of future directions for research.

Keywords: open-world, business process management, probabilistic topic modeling, process monitoring

1 Introducing open-world processes

When talking about processes in computer science and in artificial intelligence we generally refer to computational objects that, once defined, have clear-cut and predictable behaviors. The background from which such processes arise may be itself characterized by formal clarity, as is the case of the goals of a robot that through the verification offered by logical reasoning are synthesized into a process corresponding to an executable plan. Conversely, it may be of an initially murkier kind, as is the case of sequences of logs of actions or tasks performed by people with diverse roles within an organization, which can be mined and combined into explicitly defined work processes. However, in either case, the result is an object with fixed formal and computational properties, corresponding to a repeatable sequence of actions that may allow diverse degrees of flexibility, which are however known and planned for in advance.

Compare this situation with the following quote from the book *War*, a best-selling in-depth report on the current Afghan war (written by the journalist Sebastian Junger, who witnessed first-hand all the dramatic episodes described there): “the war also

diverged from the textbooks because it was fought in such axle-breaking, helicopter-crashing, spirit-killing, mind-bending terrain that few military plans survive intact even for an hour.” Without reaching such extremes, even in far more peaceful and less tragic states of affairs, there are lots of work and business processes that diverge substantially upon execution from the way they were planned, and that consequently redefine themselves dynamically according to circumstances. Such dynamic redefinitions may, more often than not, be quite radical, and hence go beyond the boundary of mere adaptation.

Clearly, these processes include all those that require real-time interactions between the agents involved and are heavily entangled with the physical world, for instance, in addition to military activities, those relating to geographical and geological exploration, logistics, energy planning. However, they include also processes where the role of the physical world is less immediate and the interaction among the participating agents is more asynchronous. This is the case with the execution of different aspects of corporate strategy (like market expansion, go-to-market plans, technology transfer, protection of intellectual property, merge-and-acquisition), the running of electoral and advertisement campaigns and financial placements.

We shall refer to processes of this kind as “open-world” by borrowing a terminology that was used in [1] to refer to the evolution of software development from a “closed-world” process to an “open-world” one. A more general way to look at these processes comes from observing that they are subject to flexible execution and are strongly influenced by external factors [2]. It should be noticed that in no way we are implying that “open-world” processes are to be considered better, or even just more relevant, than “closed-world” ones. As a matter of fact, closed-world processes are certainly easier to deal with both from the point of view of organizational models and of methods of computer support. On the other hand, open-world processes do exist. Furthermore, in a time when the traditional boundaries between organizations, institutions and countries are getting more and more friable, they are likely to increase. Hence there is room and need to increase also the support that can be derived for them from information technology.

We take the following properties to be characteristic of an open-world process:

1. First of all, it is, indeed, a process: namely it is defined as a number of steps that can be executed in a sequence in order to achieve a certain type of objective, with possible choice points subject to pre-conditions determined by its context of execution; taking a business-oriented definition, it can also be viewed as “a set of linked activities that take an input and transform it to create an output. Ideally, the transformation that occurs in the process should add value to the input and create an output that is more useful and effective to the recipient either upstream or downstream.” [3].
2. However, the process definition is open to the possibility of continuous revision, refinement, and re-interpretation due to the interaction with external agents during its execution; similarly, roles and resources in the process may need to be revised, for instance in consequence of the encounter of hitherto unknown stakeholders, or

of the impossibility to access resources that turn out to be unavailable at execution time.

3. Open-world processes of this kind are most often mission-critical for their originating organizations or institutions, that thus generally create explicit or implicit roles for decision-makers of last resort, who have the powers to redefine the process and reset roles and resources; hence the need of identifying clearly such decision makers and of extracting the decisions that they have made in order to get a grasp on the status of definition of the process and on how much it has eventually stabilized.

Point 3 defines the scope of this article, and of the future developments that can follow from it.

Our approach hinges on the tracking of increased information flows, what we call hot topics, around the execution of open-world processes, thus detecting situations where the current process execution is stressed by external factors, and an evolutionary change of its definition is therefore likely to take place. Hot topic tracking derives from the application of Probabilistic Topic Modeling (PTM) [4], a framework for the unsupervised learning of topics in collections of textual contents that has proved robust and effective in a variety of contexts. We shall illustrate an application of our approach to a specific type of external factors. In fact, we shall show how the implementation of legislation, and in particular of labor law, may affect a number of business processes where employees are among the key stakeholders, either directly, as in the case of hiring and dismissal, or indirectly, as in the case of business transfers.

The remainder of this paper is therefore structured as follows:; in Section 2 we illustrate the general principles underlying our application of PTM to the tracking of hot topics; Section 3 is the core of the paper, where we apply hot topic tracking to the monitoring of the evolution of open-world processes in the context of implementation of labor law, a very dynamic and socially “hot” sector of civil law; Section 4 outlines directions for future work and concludes the article.

2 Topic Modeling for Hot Topic Tracking

Text mining approaches based on Probabilistic Topic Modeling (PTM) are recently gaining considerable value as they allow the discovery of high-level dependencies between contents of a document corpus. Probabilistic Topic Models are statistical methods capable to handle, through unsupervised learning techniques, large volumes of unstructured data. The main purpose of these algorithms is the analysis of words in natural language texts in order to discover themes represented by sorted lists of words. For instance, Figure 1 shows 4 out of 300 topics extracted from the TASA corpus [5]. It is easy to see that words in the four topics are related to each other and can be considered as consistent *themes*. Furthermore, PTM is also able to provide topic proportions for each document, which is very useful to understand which themes a document is about. PTM-based text mining approaches aim to get the best of both worlds, by providing richly structured representations of the knowledge derived from the empirical validation of “Big Data” processing. Hence they improve both on tradi-

tional symbolic approaches, that lack data validation, and on connectionist approaches, that lack capability to represent knowledge [6].

Topic 247	Topic 5	Topic 43	Topic 56
word prob.	word prob.	word prob.	word prob.
DRUGS .069	RED .202	MIND .081	DOCTOR .074
DRUG .060	BLUE .099	THOUGHT .066	DR .063
MEDICINE .027	GREEN .096	REMEMBER .064	PATIENT .061
EFFECTS .026	YELLOW .073	MEMORY .037	HOSPITAL .049
BODY .023	WHITE .048	THINKING .030	CARE .046
MEDICINES .019	COLOR .048	PROFESSOR .028	MEDICAL .042
PAIN .016	BRIGHT .030	FELT .025	NURSE .031
PERSON .016	COLORS .029	REMEMBERED .022	PATIENTS .029
MARIJUANA .014	ORANGE .027	THOUGHTS .020	DOCTORS .028
LABEL .012	BROWN .027	FORGOTTEN .020	HEALTH .025
ALCOHOL .012	PINK .017	MOMENT .020	MEDICINE .017
DANGEROUS .011	LOOK .017	THINK .019	NURSING .017
ABUSE .009	BLACK .016	THING .016	DENTAL .015
EFFECT .009	PURPLE .015	WONDER .014	NURSES .013
KNOWN .008	CROSS .011	FORGET .012	PHYSICIAN .012
PILLS .008	COLORED .009	RECALL .012	HOSPITALS .011

Fig. 1. Example of topics extracted from the TASA corpus [5].

More technically, PTM, exploits LDA (Latent Dirichlet Allocation) [7] to automatically extract topics (concepts) from document corpora. Each topic is associated with a list of words and each document is associated with a mixture of topics. The process of topic extraction returns the probability distribution ϕ^j of the words of the document corpora for each topic j and the probability distribution θ^i of the topics for each document i . By exploiting techniques of statistical inference and sampling, these probability distributions are inferred by observing the frequency of words within documents. Figure 2 illustrates the probabilistic generative process and the statistical inference process for topic extraction.

The idea behind the use of PTM for monitoring open-world processes is to exploit the flow of information that accompanies the different steps of the process to identify situations where topics that are most closely associated with contents exchanged during process execution become densely populated. We take this as a signal that the process is going through a critical phase and that the information that accumulates around it during that period of time can contain key indicators of possible changes and re-adaptations.

To give an example on which we will return with further details later on, an inter-company process of the highest relevance is the one that governs the transfer of a business from one ownership to another. Clearly, this is an open-world business process in the sense meant in Section 1, in fact:

1. It is a set of linked activities that take an input and transform it to create an output. In this case the input is the existing ownership of a business or of a business unit, and the output is a new ownership, with a downstream recipient corresponding to the new owner, to whom the business is transferred, and an upstream recipient corresponding to the former owner, who gets the proceeds of the transaction;

2. It is open-world, since its control does not reside within the boundary of a single organization and its execution is contextually influenced by a number of factors and stakeholders which may vary and change over time.

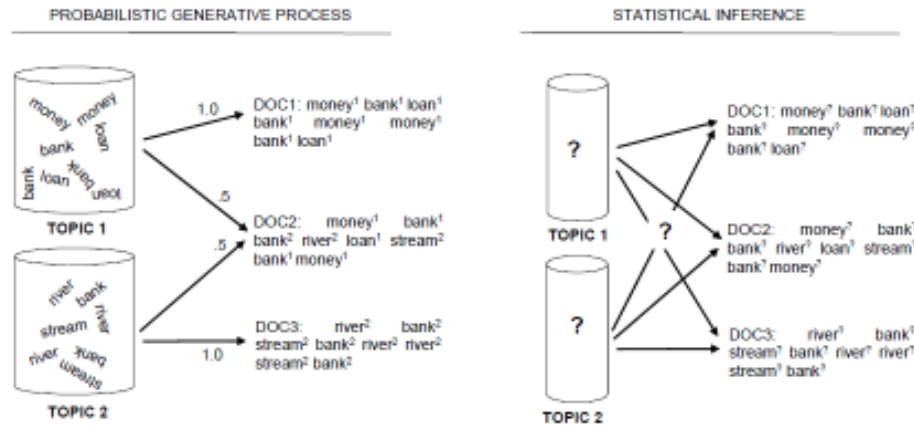


Fig. 2. From the generative process to the statistical inference [5].

For companies that operate in countries that adhere to the European Union such contextual factors and stakeholders include, respectively, the "Transfers of Undertakings Directive" of the European Union, that protects the contracts of employment of people working in transferred businesses, and the decision-makers with powers on the interpretation and application of this law. Each member state is responsible for the implementation of the Directive with respect to transfer operations of its pertinence and the rules of implementation have been generally clarified through the intervention of decision-makers of last resort, who have arbitrated conflicting interpretations during the run-in period and thus have defined the standards of implementation from then on. Considering the time generally required for a European law to stabilize within the legislative framework of the member states, and the fact that this legislation replaces since the beginning of the 2000s a previous European law in force during the last 25 years of the previous century, we can say that the Directive is reaching just now the final stage of its running-in. As we shall see, this situation is clearly reflected by the high heat of the topic most directly associated with the Directive in relatively recent times, when the Italian Court of Cassation, which is the decision-maker of last resort on this subject in Italy, made a number of judgments that interpret and define the criteria for its implementation. These documents are clustered within the topic during that interval of time, causing its heating up. After that the process of business transfer in Italy appears, as regards the Directive of Undertakings, to have reached a stable state, and this is reflected in the corresponding cooling down of the topic, as shown by the fact that in the successive time intervals the amount of content clustered within it drops very sharply.

Thus, spotting when a topic becomes "hot" and, conversely, when it "cools down", requires plotting the content carried by the topic through time. This will be illustrated in the next section.

3 Methodology

In order to substantiate the relationship between open-world business processes and hot topics we have employed LDA to classify 20.600 rulings issued by the Italian Court of Cassation in matters pertaining labor law between 2009 and 2014. This period saw several innovations of the Italian labor law, some of which attributable to the implementation of European directives in the field. In the Italian civil law system the Court of Cassation is called upon to address and validate the work of the lower courts as well as to fix the interpretation of the legislation. We can therefore expect that processes, which typically involve businesses, trade unions and workers as stakeholders, made possible by these innovations have gone through a period of adjustment solved through the deliberating activity of the Court of Cassation; such activity can in turn be reconstructed by tracking hot topics within the corpus. To achieve this, we need to plot the evolution of the measured probability of each topic against time. Let us define t as the time frame considered, $p(z|j)$ as the probability of topic z given the judgment j , $p(j|t)$ as the probability of judgment j given the time frame t and T_j as a function that associates the judgment j with the corresponding time frame. The empirical probability that an arbitrary judgment j issued at time period t was about topic z is indicated with $p(z|t)$ and it is defined in Equation 1:

$$p(z|t) = \sum_{j:T_j=t} p(z|j)p(j|t) = \frac{1}{C} \sum_{j:T_j=t} p(z|j) \quad (1)$$

Since $p(j|t)$ is the probability that the judgment is assigned to the time frame t , that term can be substituted with $\frac{1}{C}$, where C is the number of judgments in the time frame t . The function T can be parameterized to yield time intervals corresponding to one month, two months, four months, six months and one year periods.

We ran LDA setting the number of topics to extract equal to 10, 20, 50, 100 and 200 in order to find the best granularity of topics. The standard LDA model assumes that the topic structure is flat, and tries to assign a unique theme to each topic. However, if the number of topics is too big themes are split across many topics, while if the number is too little more themes can be aggregated in a single topic. While selecting the correct number of topics is still an open issue [8], an empirical analysis can be conducted to assess the topic quality. In our case domain experts, namely labor lawyers, chose the 50 topics experiment as the best candidate and specifically reviewed and graded the 50 topics set. In all 18 topics turned out to be good performers, 14 were considered noise with remaining ones being somewhat uncertain. Of the 18 good performers a further selection can be made by taking out 4 topics that are so close to other ones to correspond substantially to clones. Best performing topics have been manually labelled on the basis of the domain specific sense of topic words (Table 1).

The characteristics of a good performer, in the eyes of domain experts, can be summarily characterized in the ability to identify concepts specifically attributable to a particular legislative and / or decision-making context, e.g. “collective dismissal /

union agreement / selection criteria” or “rotation / redundancy funds / Fiat agreement”.

Table 1. Best performing topics from the 50 topics extracted.

Best performing topics	Relevant words
Transfer of business	judge (giudice), conviction (convincimento), evidence (prova), irregularity (irregolarità), company (azienda)
Collective contract	collective (collettivo), contract (contratto), agreement (accordo)
Collective dismissal	employees (dipendenti), union (sindacali), criteria (criteri), mobility (mobilità), collective (collettivo)
Work injury	injury (infortunio), liability (responsabilità), damage (danno), insurance (assicurazione)
Dismissal for just cause	contestation (contestazione), sanction (sanzione), just_cause (giusta causa), conduct (condotta), justified (giustificato)
Overtime work	compensatory_rest (riposo compensativo), damage (danno), availability (reperibilità)
Journalistic job provision	provision (prestazione), activities (attività), journalists (giornalisti), nature (natura), guarantee (garanzia)
Nature of the enterprise	cooperative (cooperativa), family (familiare), tax (tributario), administration (gestione), protection (tutela), shareholder (socio)
Fixed-term employment contracts at the Italian Post	contract (contratto), fixed-term (termine), Italian Post (Poste Italiane), damage (danno)
Impact on severance indemnities of overtime work	overtime_work (lavoro straordinario), indemnities (trattamento), compensation (compenso), national_collective_labor_contract (CCNL)
Notice and indemnity in the agency contracts	contract (contratto), agent (agente), indemnity (indennità), notice (preavviso)
Criteria of rotation in the extraordinary wages guarantee fund	extraordinary_wages_guarantee_fund (CIGS), criteria (criteri), rotation (rotazione), agreement (accordo), Fiat
European directive on transfer of undertakings	transferee (cessionario), European (europea), court_of_justice (corte di giustizia), directive (direttiva), transfer (trasferimento), seniority (anzianità)
Duties and qualifications of company directors	national_collective_labor_contract (CCNL), qualifications (mansioni), category (categoria), superiore (higher), director (dirigente)

Conversely, the characteristics of a bad performers may be multiple, some of which amendable through an improved morpho-syntactic analysis of the text, but the most frequent and endemic one resides in a composition of the topic in terms of elements too general to lead to significant identification, such as “law”, “burden”, “responsibil-

ity” and so on. We have then applied Formula [1] to monitor the trends in the topics. Topic Dismissal for just cause (the fifth from the top of Table 1) makes for an interesting and a relevant case. The theme of the topic has been in fact substantially revised by the most recent labor reform in Italy, which entered in force in June 2012, among other things by introducing relevant modifications in the open-world process related to the retaining of workers by businesses, in particular regarding so called small and medium enterprises (SMEs). We can therefore expect that immediately after that the topic would heat up. This could not be related much to the possibility to open and finalize new legal procedures on the basis of the recent legislation, which would not be possible in such a short period of time, but rather on the ability to make decisions on extant procedures by also taking into account the new norms. So it turns out to be the case: the graphic in Figure 3 shows a peak in the topic trend (probability evolution) during the second half of 2012, that on a bimonthly split can be exactly located in September 2012. After this peaking the topic progressively cools down, an indicator that the corresponding process has for the time being readjusted and stabilized.

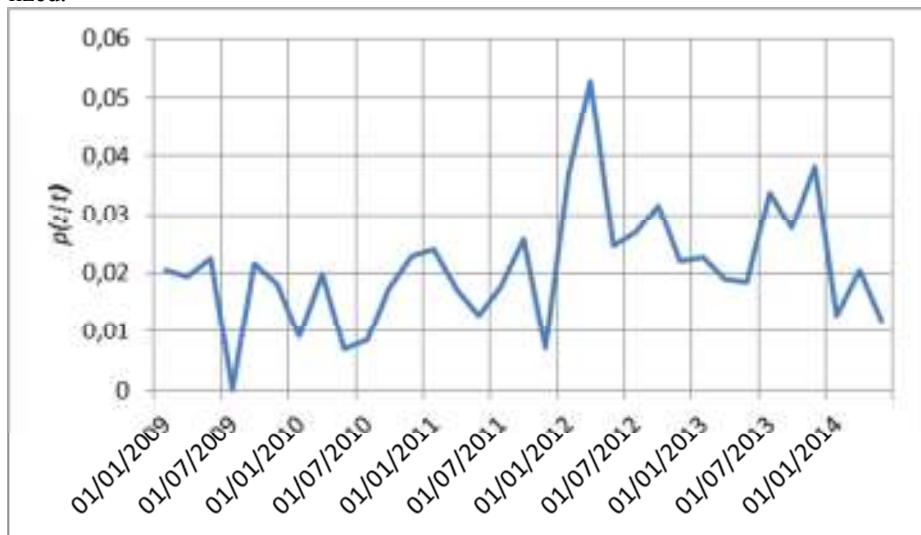


Fig. 3. “Dismissal for just cause” topic evolution.

As representative of the typical development of a hot topic, and an associated open world process, as Dismissal for just cause is, it is far from being the hottest topic among those that we have identified. In fact at the top of the “hit parade” of hot topics we find the second last item from Table 1, namely European directive on undertakings. We can compare how far hotter it is with respect to Dismissal for just cause by plotting the trends of the two topics one against the other as in the graphic in Figure 4, where we can also notice that the latter topic peaks up at its highest probability value during the second half of 2011 and then resurges sharply yet again, even if without reaching the previous heights, for a longer period encompassing most of the second half of 2012 and of the first half of 2013. Indeed, it is in this period that the Italian

Court of Cassation issues a number of rulings that have become fundamental benchmarks for the implementation in the Italian context of the European directive on transfer of a business (or of a business unit). We can also ask ourselves why European directive on undertakings is so much hotter as a topic than Dismissal for just cause. This may find a plausible answer in the fact that the scope of European directive on undertakings, that concerns companies of all sizes, and touches an issue of foremost importance (sometimes decisive for the fate of thousands of workers), is so much wider than the changes affecting the scope of Dismissal for just cause, where the actors mostly concerned are SMEs and the dealt cases are about individual workers. As an example of a mid-flyer we can find topic Overtime work, dealing with theme of compensation for overtime work, a subjects that is well-known and established, but given its numerous interpretations and social relevance, is bound to heat up from time to time, with the Court of Cassation acting an actor of arbitration and regulation for the diverse options open in the execution of the processes.

Finally, we can see in Figure 4 these three topics against two topics that have been deemed as non-performing by the domain experts. As can be expected, the topics develop in a very flat way, by spreading evenly among all processes, and thus lack the capability of highlighting major turning points in their definition and implementation.

As a note on related work, a somewhat similar equation has been applied in [9] with a different purpose, namely the statistical and quantitative reconstruction of the history of ideas in a variety of scientific areas. Indeed, the focus of [9] is a case study on the evolution of research directions in computational linguistics through the topic-based analysis of 12,500 articles published in major international conferences in the field between 1980 and 2005. It is interesting to note that the trends reported in this work show characteristics significantly different from those reported here. In fact, the graphics contain mostly gentle slopes as opposed to the abrupt peaks with steep ascents and descents characterizing, albeit with varying levels of abruptness, our graphics. This difference has a variety of reasons, the most obvious and paramount of which is that the analyzed documents are, in this case, typically relevant for the advancement of a scientific discipline, a phenomenon that is indeed distributed over time, but with none of the characteristics of processes aimed at attaining specific goals, with which the documents analyzed in our case are associated. Thus, the gradual emerging and waning of ideas, caught by such softer uphill and downhill, is precisely what we expect, in contrast to the sharp phases of adjustment, as required by the practical needs of the moment, that characterize our topics. [10] do address the notion of hot topic in a vein very similar to ours, but their formal and computational treatment falls completely outside PTM and LDA, and in fact is term-oriented rather than topic-oriented. Thus, it does not appear suitable for the spotting of hot topics from large content corpora which is our purpose here, while it may be particularly suitable for their identification in the context of short texts, such as the tweets or the threads of social networks.

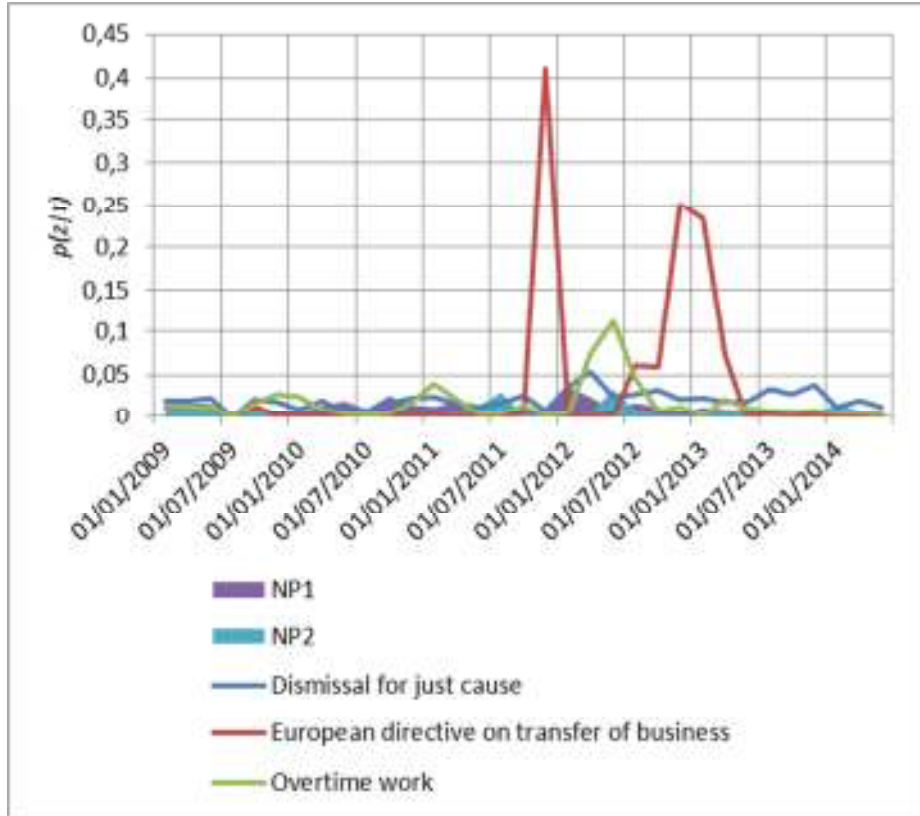


Fig. 4. Topic evolution of significant topics compared to non-performing topics (NP).

4 Conclusions

We have introduced the notion of open-world process in order to capture large processes that involve multiple organizations, and are influenced by a number of contextual factors and stakeholders. Furthermore, we have introduced the notion of hot topic so as to provide a computationally effective way to track context evolution around open-world processes in terms of the information that flows into such processes at various degrees of density over time. Hot topics are themselves an application of statistical inference in the form of PTM, and thus are firmly rooted into empirical evidence, without sacrificing high level representations of the inferred meanings. Hence they show promise to be effectively combined with existing high-level representations of business processes. There are a number of research directions that can and, in order to obtain useful and concrete results, must be pursued to evolve this initial contribution. A most immediate one is to carve out a framework for Open-world Business Process Management within the wider field of Business Process Management [11], the discipline that encompasses the established computational framework for the

management of closed-world processes, namely Workflow Management. One clear direction to achieve this is to combine our statistical approach to the monitoring of information flows in open-world processes with a general framework for the monitoring of events over time in global systems, like, for instance, the reactive version of the Event Calculus proposed in [12]. Once verified the heating up of a topic over a defined interval of time such a calculus may then trigger the inspection of the pertinent contents, and the consequent retuning of the closed-world processes in the participating organizations that are synchronized around the relevant open-world process, through adaptive technologies like adaptive workflow management [13], [14] and case handling management [15].

Another crucial step is to survey and consequently map the existing open-world processes from the informal sources through which they are currently documented into a more rigorous notation. Given the inherent definitional fluidity of open-world processes, in place of the formal notations commonly used for closed-world processes, like Petri Nets and Workflow Nets, it may be preferable to use one aimed at handling incomplete information and soft constraints, such as the Generalized Process Structure Grammars (GPSGs) presented in [16]. GPSGs that encode open-world processes could in fact include rules containing symbols as names for topics generated through LDA/PTM, thus acting as interfaces between process definitions and contextual information flows. Another approach that could be similarly adopted, being itself based on a declarative formalism for the definition of flexible processes, is the one presented in [2].

As far as the topics are concerned, there are further possible constructions that can help us to gain insight about the processes they are associated with. In particular, one possibility we plan to explore is the generation of links among semantically related content objects clustered within the topics, following the methodology presented in [17]. In the context of the specific case study presented here, based on the implementation of labor law, this would allow us to rate the relevance of judgments on the basis of how many other judgments make reference to it, this being a clear case of semantic proximity. However, given that semantic proximity between content objects is itself probabilistically computed, other less obvious relationships would emerge. In this way, we could gain access not just to hot topics, but also to hot objects.

5 Bibliography

1. L. Baresi, E. Di Nitto e C. Ghezzi, «Toward Open-World Software: Issues and Challenges,» *IEEE Computer* 39(10), pp. 36-43, 2006.
2. M. Pesic e W. M. van der Aalst, «A Declarative Approach for Flexible Business Processes Management,» in *Business Process Management Workshops*, 2006.
3. H. J. Johansson, P. McHugh, J. Pendlebury e W. A. Wheeler, *Business Process Reengineering: Breakpoint Strategies for Market Dominance*, John Wiley & Sons, 1993.
4. D. M. Blei, «Probabilistic Topic Models,» *Commun. ACM* 55(4), pp. 77-84, 2012.
5. M. Steyvers e T. Griffiths, «Probabilistic Topic Models,» in *Handbook of Latent Semantic Analysis*, 2007, pp. 424--440.

6. J. B. Tenenbaum, C. Kemp, T. L. Griffiths e N. D. Goodman, «How to grow a mind: statistics, structure, and abstraction,» *Science* 331 (6022), pp. 1279-1285, 2011.
7. D. M. Blei, A. Y. Ng e M. Jordan, «Latent Dirichlet Allocation,» *The Journal of Machine Learning Research*, pp. 993--1022, 2003.
8. E. H. Ramirez, R. Brena, D. Magatti e F. Stella, «Topic Model Validation,» *Neurocomputing*, pp. 125-133, 2012.
9. D. Hall, D. Jurafsky e C. D. Manning, «Studying the History of Ideas Using Topic Models,» in *EMNLP*, 2008.
10. K.-Y. Chen, L. Luesukprasert e S.-c. T. Chou, «Hot Topic Extraction Based on Timeline Analysis and Multidimensional Sentence Modeling,» *IEEE Trans. Knowl. Data Eng.* 19(8), pp. 1016-1025, 2007.
11. W. M. P. van der Aalst, A. H. M. ter Hofstede e M. Weske, «Business Process Management: A Survey.,» in *Business Process Management*, 2003.
12. S. Bragaglia, F. Chesani, P. Mello, M. Montali e P. Torroni, «Reactive Event Calculus for Monitoring Global Computing Applications,» in *Logic Programs, Norms and Action*, 2012.
13. U. Borghoff, P. Bottoni, P. Mussio e R. Pareschi, «Reflective Agents for Adaptive Workflows,» in *Second International Conference on the Practical Application of Intelligent Agents and Multiagent Technology*, London, 1997.
14. M. Leitner, S. Rinderle-Ma e J. Mangler, «AW-RBAC: Access Control in Adaptive Workflow Systems,» in *ARES*, 2011.
15. W. M. P. van der Aalst, M. Weske e D. Grünbauer, «Case handling: a new paradigm for business process support,» *Data & Knowledge Engineering* 53 , pp. 129-162, 2005.
16. N. S. Glance, D. Pagani e R. Pareschi, «Generalized Process Structure Grammars for Flexible Representations of Work,» in *CSCW*, 1996.
17. M. Rossetti, R. Pareschi, F. Stella e F. Arcelli , «Integrating Concepts and Knowledge in Large Content Networks,» *New Generation Comput.* 32(3-4), pp. 309-330, 2014.
18. T. L. Griffiths e M. Steyvers, «Finding scientific topics,» *Proceedings of the National academy of Sciences of the United States of America*, pp. 5228-5235, 2004.

Using Semantic Lifting for Improving Educational Process Models Discovery and Analysis

Awatef Hicheur Cairns¹, Joseph Assu Ondo¹, Billel Gueni¹, Mehdi Fhima¹, Marcel Schwarcfeld¹, Christian Joubert¹, Nasser Khelifa²,

¹ALTRAN Research, ² ALTRAN Institute
Vélizy-Villacoublay, France

{awatef.hicheurcairns, joseph.assu, billel.gueni,
mehdi.fhima, marcel.schwarcfeld, christian.joubert,
nasser.khelifa}@altran.com

Abstract. Educational process mining is an emerging field in the educational data mining (EDM) discipline, concerned with discovering, analyzing, and improving educational processes based on information hidden in datasets and logs. These data are recorded by educational systems in different forms and at different levels of granularity. Often, process discovery and analysis techniques applied in the educational field have relied exclusively on the syntax of labels in databases. Such techniques are very sensitive to data heterogeneity, label-name variation and their frequent changes. Consequently, large educational process models are discovered without any hierarchy or structuring. In this paper we show how by linking labels in event logs to their underlying semantics, we can bring educational processes discovery to the conceptual level. In this way, more accurate and compact educational processes can be mined and analyzed at different levels of abstraction. We have tested this approach using the process mining Framework ProM 5.2.

Keywords: Semantic Process Mining, Educational Process Mining, Ontology, Semantic Matching, ProM.

1 Introduction

Nowadays, education and training centers promote personalized curriculums where students are free to choose the skills they want to develop (from beginner to specialist), the way they want to learn (theoretical or practical aspects) and the time they want to spend. This tendency is reinforced by the emergence of "e-learning" which represents an increasing proportion of the in-company trainings. Educational systems support a large volume of data, coming from multiple sources and stored in various formats and at different granularity levels [6], [16]. These data can be exploited by instructors to understand students' learning habits, the factors influencing their performance and their target skills. To answer these questions, there is an increasing research interest in using process mining in education [6],[10], [15], [16]. The idea of process mining [1]

is to discover, monitor and improve real processes (i.e., not assumed processes) by extracting knowledge from event logs (recorded by an information system). However, the proposed approaches for process models extraction in the education field are somewhat limited because they rely on classical process mining techniques which are purely syntax oriented i.e. based on the labels in event logs [2]. For instance, we have encountered a massive professional training dataset of a worldwide consulting company where depending on the country and the region involved different names were used for the same training. So, the actual semantics behind the trainings' labels remain in the head of education management people (e.g. teachers, carrier advisors, etc.) who have to interpret them. To handle this question, semantic annotations on event logs could be used to prevent such interpretation efforts [2], [3]. To benefit from the actual semantics behind these labels, *semantic process mining* techniques were introduced in [2], [3], [4], leveraging mining and analysis techniques to the conceptual level. In this paper, we show how semantic process mining ideas may help to discover simplified educational process models and to extract more knowledge about their properties. For the first time, to our knowledge, a professional training dataset of a consulting company is taken as a case study to extract and analyze training paths annotated with semantic information. Also, we propose a (semi)automatic procedure used to associate semantics to training labels. The remainder of this paper is organized as follows. Section 2 summaries educational process mining techniques. Section 3 presents the semantic process mining core idea. Section 4 explains our approach to extract educational process models annotated with semantic information. Finally, section 5 concludes the paper.

2 Process Mining in the Educational Field

Process mining is a relatively new technology which emerged from the information technology and management science [1]. It focuses on the development of automated techniques to extract process-related knowledge from event logs. An *event* log corresponds to a set of process *instances* (i.e. *traces*) following a business process. Each recorded event refers to an *activity* and is related to a particular process instance. An event can have a *timestamp* and a *performer* (i.e. a person or a device executing or initiating an activity). *Educational Process Mining* (EPM) refers to the application of process mining techniques in the education domain [16]. Educational event logs may include students' registration procedures, student's examination traces or activity logs in e-learning environments. The three major types of process mining techniques are (cf. Fig. 1): *Process model discovery* takes an event log and produces a complete process model able to reproduce the behavior observed in this log. *Conformance checking* aims at monitoring deviations between observed behaviors in event logs and process models or predefined business rules and constraints. *Process model extension* aims to improve a given process model based on information (e.g., time, performance, case attributes, decision rules...etc.) extracted from an event log related to the same process. Regarding available process mining tools, the ProM Framework is the most complete and powerful one aimed at process analysis and discovery from all perspectives (process, organizational and case perspective) [8]. It is implemented as an open-source Java application with an extendable pluggable architecture.

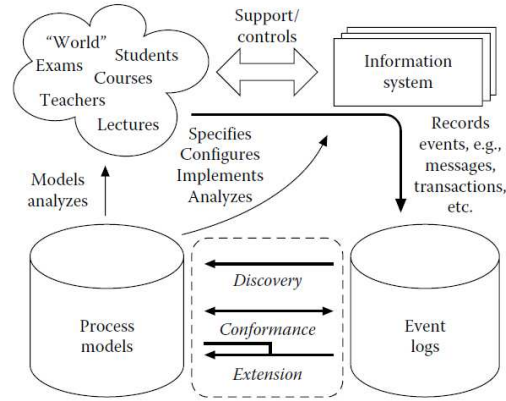


Fig 1. Process mining concepts

ProM supports a wide range of techniques for process discovery, conformance analysis and model extension, as well as many other tools like conversion, import and export plug-ins. The de facto standard for storing and exchanging events logs are the MXML (Mining eXtensible Markup Language) format or more recently the XES (eXtensible Event Stream) format. In practice, however, ProM presents certain issues of flexibility and scalability which limit its effectiveness in handling large logs from complex industrial applications [13]. We may get over these limitations by using the service oriented architecture of the ProM 6 framework. Theoretically, such architecture may allow the distribution of ProM's plugins over multiple computers (e.g., grid computing). We are recently testing such a construction in the development of an interactive and distributed platform tailored for educational process discovery and analysis. Let us note that, lately, educational process mining has emerged as a promising and active research field [6], [15], [16]. However, the application of process discovery techniques presents some challenges given the huge volume and the traces' heterogeneity often encountered in educational datasets. In fact, when analyzing event logs containing a lot of distinct traces, traditional process discovery techniques generate highly complex models (i.e. spaghetti models) [13]. In this case, the adoption of filtering, abstraction or clustering techniques may help reduce the complexity of the discovered process models [14], [17]. For instance, a clustering technique was proposed in [6] to improve both the performance and readability of the mined students' behavior models in the context of e-learning. In our previous work [10], we proposed a two-step clustering approach for partitioning training processes depending on an employability indicator. We think that *semantic process mining* techniques seem to be a promising area to explore in order to handle the issue of traces' heterogeneity and so to extract simplified process models.

3 Semantic Process Mining

The *semantic process mining* techniques, introduced in [2], [3] aim to analyze and extract process-related knowledge from event logs, at the conceptual (semantic) level

[4]. The challenges for mining and monitoring processes from a semantics perspective have been studied in the context of the European project SUPER [9]. The concept of semantic log purging was proposed in [12], taking a case study in the higher education domain. In [5], the authors proposed a combination of standard process mining techniques with semantic lifting procedures on the event logs in order to mine more precise process models. The core idea of semantic process mining is to explicitly annotate elements in event logs with the *concepts* that they represent. These concepts are formalized in generic or domain specific ontologies. Hence, semantic process mining techniques are built on the following three basic elements: *ontologies*, *ontology reasoners*, and *references* from elements in logs/models to concepts in ontologies [2]. First, ontologies define and formalize a set of concepts shared by (a group of) people to refer to things in the world and the relationships among these concepts. Second, the reasoner provides reasoning over the ontologies in order to derive new knowledge, e.g., subsumption, equivalence, etc. Finally, the references associate meanings to labels (i.e., strings) in event logs and/or models by pointing to concepts defined in ontologies. The discovery, conformance checking, and extension techniques rely on subsumption relations induced by these ontologies to raise the level of abstraction from the syntactical level to the semantical level. Thus, these techniques can be applied without requiring any modification of models or logs if the elements in different logs and models link to the same concepts (or super/sub concepts of these concepts). Let us note that all semantic plug-ins developed in ProM are based on the following concrete formats for the basic building blocks: Event logs are in the SA-MXML (i.e. Semantic Annotated Mining eXtensible Markup Language) file format. SA-MXML is a semantically annotated version of the MXML format which incorporates the model references (between elements in logs and concepts in ontologies). Ontologies are defined in WSMML (Web Service Modeling Language) [7], [11]. The WSMML 2 Reasoner Framework [18] is used to perform all the necessary reasoning over the ontologies.

4 Case Study: Leveraging Educational Process Mining Techniques at the Semantic Level

Our motivating example is based on real-world training databases from a worldwide consulting company. This company has around 6 000 employees that are free, during their careers, to take different trainings aligned with their profiles. These trainings are provided by internal or external organizations. The data collected for analysis includes the employees' profiles (demographics data), their careers (i.e. the jobs/missions they did) and their training paths (the set of trainings taken during the past three years) (cf. Table 1). In what follows, we apply a process model discovery algorithm (e.g. the heuristic miner [8]) on a fragment of the training event log (cf. table 1), containing 1000 traces, 2419 events and 280 originators. We can see that the obtained result is an unreadable spaghetti like process model (cf. Fig. 2). This result can be explained by the heterogeneity in employees' training paths and the great number of different trainings' labels. Let us note that depending on the organization, the country and the region involved, different labels (i.e. string) were used for the same training. Moreover, some training courses can be seen as special cases of other trainings. For instance, the trainings "*Collective English*", "*Collective Face to Face*"

English”, “*English In Group*” are in fact the same training which is given different names following data sources. Moreover “*Collective Face to Face English*” is a variant of “*Face to Face English*”, which is a special type of the “*English*” training.

Table 1. Example of an educational event log

Matricul	Profil	Training_Id	Training_Label	Training_Orga_Id	StartDate	EndDate
7	CONSULTANT	Tr 850	EXCEL ELEARNING	Org 13	11/07/2011	31/12/2011
8	CONSULTANT	Tr 769	QF TEST	Org 135	26/04/2011	28/04/2011
9	CONSULTANT	Tr 252	INTERCULTURAL WORKING RELATONS : INDIA	Org 135	01/07/2011	01/07/2011
10	CONSULTANT	Tr 260	SELENIUM	Org 135	25/10/2011	26/10/2011
11	CONSULTANT	Tr 812	UML FUNCTIONAL ANALYSIS	Org 135	24/10/2011	27/10/2011
12	CONSULTANT	Tr 774	DESIGN PATTERNS AND APPLICATION C++	Org 135	08/12/2011	09/12/2011
13	CONSULTANT	Tr 1923	SQL BASIC	Org 135	03/04/2012	05/04/2012
14	CONSULTANT	Tr 813	C++ ADVANCED	Org 135	04/04/2012	06/04/2012
15	CONSULTANT	Tr 2014	XML BASIC AND XPATH	Org 135	10/04/2012	11/04/2012
14	CONSULTANT	Tr 1282	DESIGN PATTERNS AND APPLICATION IN C++	Org 135	13/09/2012	14/09/2012
...			

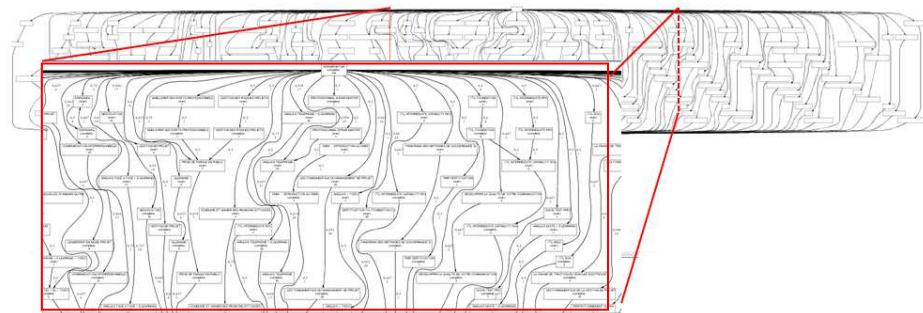


Fig 2. Fragment of a spaghetti process describing all trainings followed by the consulting company’s employees during the last three years. The process model was extracted using the Heuristic Miner plug-in of ProM.

To handle this issue, we need to link different trainings which are variants or synonyms of the same training to a unique concept in a *training ontology*. Usually, there are two ways to achieve this. We can manually create all the necessary ontologies and annotate the necessary elements in educational event logs with ontologies’ concepts. It is also possible to use tools to (semi)automatically discover ontologies based on the elements in these logs [4]. The discovered ontologies can be manually improved in a second step. Let us note that semantic process mining tools can also play a role in ontologies’ extraction and enhancement from event logs. The ontology depicted in Fig. 3 is used to formalize the concepts for trainings in our example. It contains 42 concepts and 129 instances. We built this ontology manually taking as starting point the semantic description of trainings provided in training organizations’ catalogues. We distinguished five super-concepts related to the training domain: *Communication*, *Staff Management*, *Project Management*, *Audit and Control*, *Information Technologies*.

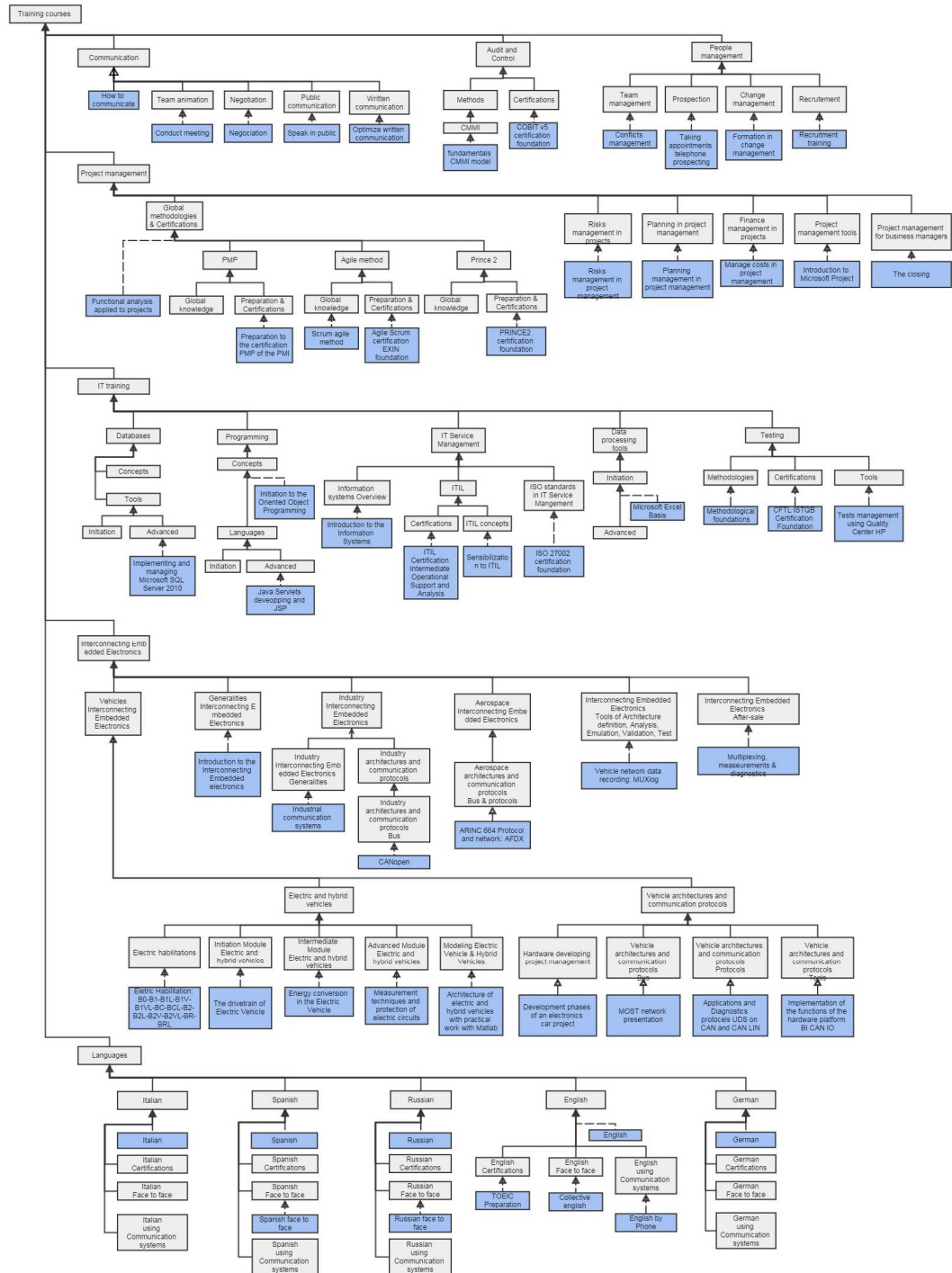


Fig 3. Fragment of the “Training ontology”: only some instances (i.e. training labels) are represented

These concepts are subdivided into sub-concepts which are in their turn subdivided into lower sub-concepts (cf. Fig 3). Trainings' labels are the instances of this ontology and each label is associated to one concept or sub-concept. To simplify the ontology depicted in Fig 3, we only represented one instance (training label) per concept. We used the tool WSMT (Web Service Modeling Toolkit) to implement the training ontology in the WSML format since it is supported by the ProM 5.2 framework. Moreover, the semantic process mining plug-ins existing in ProM 5.2 expect log elements to be connected with process ontologies (i.e., to be in the SA-MXML logging format). So to enrich the educational log of our example with semantic annotations from the *Training Ontology*, we implement a conversion plug-in in ProM 5.2. The latter takes as input the original *educational log* (in MXML format) and the *Training Ontology* (in WSML format) and produces the corresponding semantically annotated event log (in SA-MXML format).

4.1 Semantic Matching Between Training Labels and Concepts

In order to help end users in the comprehension of the underlying semantics of training courses, we develop a (semi)automatic procedure, which can be used to associate a concept (of the training ontology) to a training label. The association used is based on the importance of the words in a label or in a concept. We assume that each word of a label L plays the same semantic role and hence has the same importance as well as the other words constituting L . We also suppose that at least one of the words characterizing a concept, or one of their synonyms, appears in all the labels associated to it. Therefore, there is an intersection between the set of the words of a label and the set of the words characterizing its associated concept. To build our technique we develop the following modelling: consider $W = \{w_1, \dots, w_n\}$ a set of words, we consider a training label TL_i as succession of w_j , noted $TL_i = w_1 \mathbf{b} + \dots \mathbf{b} + w_m$, where $w_j \in W$ and the symbol \mathbf{b} represents blanks and all articles, pronouns, etc. For instance the label "*Introduction to Information Systems*" contains the set of words $W = \{\text{Introduction, Information, Systems, Management}\}$ separated with three blanks and the preposition 'to'. We consider L_i the set of the words that contains TL_i , so $L_i = \{w_1, \dots, w_m\}$ and in our case we assume that $card(L_i)$ represents the length of the label TL_i (we note $Len(TL_i)$), for example $Len(\text{Introduction to Information Systems}) = 3$. We also consider $C_j = \{w'_1, \dots, w'_k\}$ as the set of the words characterizing a concept C_j .

Word importance: is a metric, or a weight, reflecting the importance of a word in a label according to our hypothesis given below. As each word plays the same role in a label we compute its importance wp as follow: $wp(w) = 1 / Len(TL)$ where $w \in L$. for the label $TL = \text{"Management in Information Systems"}$, $Len(TL)=3$ and $wp(\text{Management})=1/3$. This wp reflects clearly the relation between the length of a label and the importance of its word. A small label, like ones using only one or two words, gives a great semantic importance to its word that are considered like keys, whereas long labels use lot of words for their description giving its words a small semantic role.

Word concept weight: the weight of a word w in a concept C , noted $cw(w)$, corresponds to the sum of all word importance of w , or one of its synonyms, in all the labels associated to the concept C : $cw(w) = \sum wp_{TL_i}(x)$, where $i \in \{1, \dots, h\}$ and TL_i is associated to C . For instance, consider the concept characterized by the following

words (“management”, “project”). If “management” appears three times in the labels with the following wp : $\frac{1}{2}$, $\frac{1}{2}$ and $\frac{1}{3}$ therefore $cw(\text{“management”}) = \frac{1}{2} + \frac{1}{2} + \frac{1}{3} = 1.3$. This metric establishes a monotone relation between the frequency of the word in the labels and its importance, and it is clear that more a word is used, more it is important and more it will be used to characterize a concept.

Concept matching: to generate automatically the concept C associated to a label TL we create first a word weight table as follow:

1. We compute the set of all the words of all the labels contained in the training catalogue. We note this set as LW .
2. We create a matrix $M = (a_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq m)$ $a_{i,j}$ is the $wp(i)$ in the label j , $n = \text{card}(LW)$ and m is the number of all training Labels.
3. For each word w in LW , we sum its $wp(w)$ computed in the previous step and we store the result in the returned table.

After constructing this table, for a label TL we compute the semantic intersection between L and C as follow: $L \cap C = \{w_j, w_j \in L \wedge w_j \in C\}$. $w_j \in C$ means that w_j or a synonym of it is included in C . Then we compute the score of matching between L and C , noted $SC(L, C)$ as the sum of the *concept weight* of each element of $L \cap C$. We repeat this operation for all the concepts we have and then we associate L with the concept having the high score. If we have the concepts C_1, \dots, C_n then L will be associated to C if $SC(L, C) = \text{Max}(SC(L, C_n))$. The semantic importance we use in our matching is simplified compared with approaches doing deep semantic analysis using sophisticated techniques because we do a significant human effort to define the Ontology with different level, and we stress on the concepts of the level 2 to enrich them with words that are generally and mostly used to define the labels associated to each concept of this level. We remark that if we have two or more concepts having the same $\text{Max}(SC(L, C_n))$ we infer a conflict and in this case we need a user’s intervention to choose what concept to associate to the label. We have tested this matching technique on Altran catalogue containing 128 labels and 35 concepts. Fig 4 depicts the obtained results. Let us note that in these tests we have identified some cases where we have not identified matching between labels and concepts.

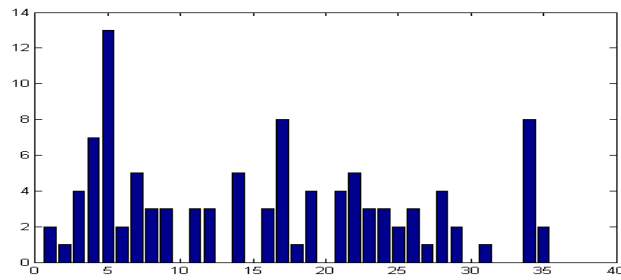


Fig 4. The number of labels (ordinate) associated to each one of the 35 concepts (absciss) of our case study

This is due to the use of some abbreviations that are hard to decrypt. In these tests, concepts contain only words that we find in labels and we do not need in this case to

search synonyms. We plan in the future to use a dictionary in order to enhance the identification of synonyms.

4.2 Educational Process Models Mining at the Conceptual Level

After constructing a semantically annotated educational log, we specify the level of abstraction (i.e. concepts in the training ontology) used as a base for the mining and the analysis of training processes. To achieve this, we use the *filter* plug-in “*Ontology Abstract Filter*” implemented in Prom 5.2, which allows us to choose the required level of abstraction [8]. The *Ontology Abstract Filter* plug-in takes as input a semantically annotated event log (in SA-MXML format) and produces as output another event log where the names of tasks (i.e. trainings) are replaced by the names of the chosen concepts. The produced log can also be exported as an SA-MXML log. After this step, we may apply a control-flow mining algorithm (e.g. the *Heuristic Miner* plug-in) to extract the educational process model relaying on the concepts chosen in the previous step. We may choose concepts at different level of abstractions. When we use only the concepts at level 2 of the *Training Ontology* tree (i.e., the concepts “Communication”, “Language”, “Testing”, “Audit_And_Control”, “IT_Service_Management”...etc.”), a process model like the one in Fig. 5 could be discovered.

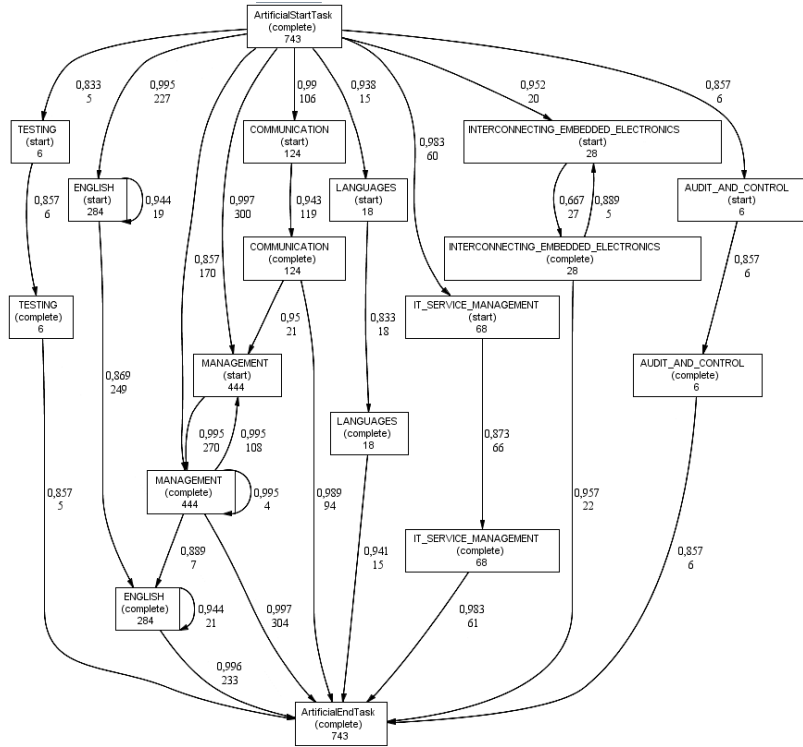


Fig 5. Training process model mined using the heuristic miner plug-in where only the concepts at the level 2 of the tree for the ontology “TrainingOntology” (cf. Fig.3) are considered.

It contains 18 events (nodes) and 30 arcs which is more compact than the model extracted before the semantic abstraction (cf. Fig 2). Let us note that during the abstraction phase we deliberately replace the labels of the different kind of English trainings by their concept at level 1 (i.e. *English*). We can see that the mined model in this case is more compact (i.e., has a higher abstraction level) than the one in Fig 2. In this model we can see that trainings associated to the concept “*Management*” are taken 444 times. Also, there are seven trainees who took an “*English*” training after a “*Management*” training. The frequency associated to this relation in the educational log is 0,889.

4.3 Educational Process Analysis at the Conceptual Level

In our case study, process mining advantages are not limited to the discovery of employees’ training processes. In fact, training advisors and directors of training organizations often need to check (off-line or on-line) whether trainees’ paths conform to established career paths, trainings’ prerequisites or business rules. The semantic LTL checker plug-in of ProM 5.2 is the perfect tool for auditing educational processes at the conceptual level [2]. This tool can be used to verify the same formula (e.g. generic formula such as prerequisite) on a set of different event logs as long as the arguments of this formula and the elements in these logs link the same concepts (or super/sub concepts of these concepts). There is a set of predefined formulas in the semantic LTL model checker plug-in. It is also possible to tailor the semantic LTL checker plug-in to express specific types of constraints encountered in the educational domain [16]. All these properties can be easily coded using the LTL language and imported into the user interface of the plug-in. In what follows we want to check if the rule “A *Project Management* training must be taken before a *Project Management Professional Certification (PMP)* can be taken” was always respected (prerequisite check). We define this property in a LTL file as follows:

```
formula c2_is_a_prerequisite_of_c1 ( c1 : ate.WorkflowModelElement, c2 :
ate.WorkflowModelElement) :=
{
  <h2> Is the training C2 a prerequisite for the training C1? </h2>
  ( <> (activity == c2) /\ (activity != c2 _U activity == c1) ) ;
}
```

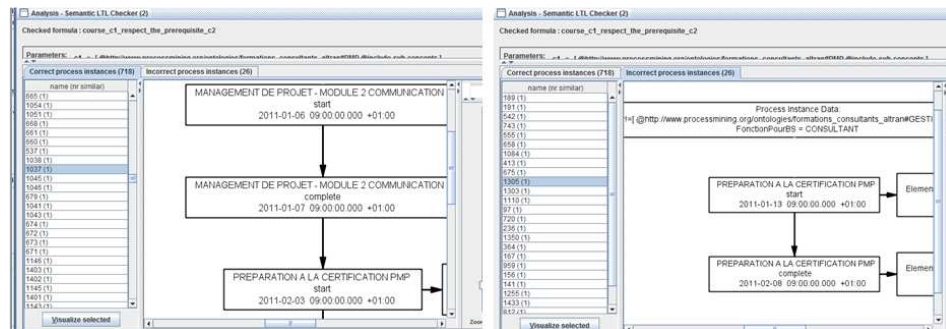


Fig 6. The results returned by the semantic LTL Checker plug-in while verifying the PMP prerequisite

Fig 6 shows the result displayed when this property is checked. We can see that there are 26 trainees who took the *PMP* training while they didn't take the *Project Management* training before (i.e., incorrect case instances). There are also 718 trainees that satisfy this property (i.e. they took the "*PMP*" training after a "*Project Management*" training).

5 Conclusion

In this paper we showed how by associating semantic annotations to educational event logs, more accurate and compact educational processes can be extracted and analyzed at different levels of abstraction. Also we developed a semantic matching procedure allowing to link training labels to the right concepts of a training ontology, in a (semi)automatic way. In future works, we will investigate how concepts from ontologies can be associated to training providers. We can then benefit from these semantic annotations in mining social networks and organizational models between training providers [1], [10], at the conceptual level. We plan also to conduct a case study in an on-line education setting that would illustrate the benefit of process mining approaches, at the syntactic and semantic levels, to mine and understand students' behaviors. Another important step in our works is to develop new clustering and classification techniques which take into account semantic annotations on event logs [14], [17]. For instance, trace clustering techniques [14] can be extended to partition event logs depending on trace similarities at the conceptual level. To implement our approach, we are currently developing an interactive and distributed platform tailored for educational process discovery and analysis. This platform will allow different education centers and institutions to load their data and access advanced data mining and process mining services [10]. Moreover, in order to optimize and enhance platform response time, our platform will allow distributing heavy analysis computations on many processing nodes.

Acknowledgments. This ongoing work is being carried out by Altran Research and Altran Institute within the context of the PHIDIAS project.

References

1. Aalst, W. M. P., and al.: Process Mining Manifesto, In BPM 2011 Workshops Part I, LNBIP 99, pp. 169-194 (2012).
2. Alves de Medeiros, A.K., van der Aalst, W.M.P., Pedrinaci, C: Semantic Process Mining Tools: Core Building Blocks. In 16th European Conference on Information Systems, pp. 1953-1964. Galway, Ireland (2008).
3. Alves de Medeiros, A. K., . van der Aalst, W. M. P: Process Mining towards Semantics. Advances in Web Semantics I, LNCS 4891, pp 35-80 (2009).
4. Alves de Medeiros, A. K. and al: An Outlook on Semantic Business Process Mining and Monitoring. In R. Meersman, Z. Tari & P. Herrero (Eds.), The Confederated

- International Conferences On the Move to Meaningful Internet Systems, LNCS, vol. 4806, pp. 1244-1255, Springer-Verlag (2007).
5. Azzini, A., Braghin, C., Damiani, E., Zavatarelli, F: Using Semantic Lifting for improving Process Mining: a Data Loss Prevention System case study. In the 3rd International Symposium on Data-driven Process Discovery and Analysis, CEUR-WS.org, pp. 62-73 (2013).
 6. Bogarín, A., Romero, C., Cerezo, R., Sánchez-Santillán, M: Clustering for improving educational process mining. In Proceedings of the Fourth International Conference on Learning Analytics And Knowledge. ACM, New York, NY, USA, pp. 11-15 (2014).
 7. de Bruijn, J., Lausen, H., Polleres, A., Fensel, D.: The Web Service Modeling Language WSMML: An Overview. In Y. Sure and J. Domingue, editors, ESWC, LNCS, vol. 4011, pp. 590-604. Springer (2006).
 8. van Dongen, B. F., de Medeiros, A. K. A., Verbeek, H. M. W., Weijters, A. van der Aalst, W. M. P: The Prom Framework: a New Era in Process Mining Tool Support. In ICATPN'05, Gianfranco Ciardo and Philippe Darondeau (Eds.). LNCS, vol. 3536, pp. 444-454, Springer-Verlag, Heidelberg (2005).
 9. European Project SUPER - Semantics Utilised for Process Management withing and between Enterprises. <http://www.ip-super.org/>
 10. Hicheur Cairns A. and al: Custom-Designed Professional Training Contents and Curriculums through Educational Process Mining. In The Fourth International Conference on Advances in Information Mining and Management, pp 53-58 (2014).
 11. Lausen, H., de Bruijn, J., Polleres, A., Fensel, D: The WSMML Rule Languages for the Semantic Web. W3C Workshop on Rule Languages for Interoperability, W3C (2005).
 12. Ly, L.T, Indiono, C., Mangler, J., Rinderle-Ma, S.: Data transformation and semantic log purging for process mining. In Proceedings of the 24th international conference on Advanced Information Systems Engineering, J. Ralyté and al. (Eds.).LNCS, vol. 7328, Springer-Verlag (2012)
 13. Reichert, M.: Visualizing Large Business Process Models: Challenges, Techniques, Applications. In 1st Int'l Workshop on Theory and Applications of Process Visualization Presented at the BPM 2012, LNBIP, vol. 132, pp. 725-736, Springer-Verlag (2013).
 14. Song, M., Günther, C.W., van der Aalst, W.M.P.: Trace Clustering in Process Mining. In: Ardagna, D., Mecella, M., Yang, J. (eds.) BPM 2008. LNBIP, vol. 17, Springer, Heidelberg, pp. 109–120 (2009).
 15. Trčka, N., Pechenizkiy, M: From Local Patterns to Global Models: Towards Domain Driven Educational Process Mining. In Proc. 9th International Conference on Intelligent Systems Design and Applications, pp. 1114-1119. IEEE Computer Society (2009).
 16. Trčka, N., Pechenizkiy, M., van der Aalst, W.: Process Mining from Educational Data (Chapter 9). In Handbook of Educational Data Mining. pp. 123-142. CRC Press (2010).
 17. Veiga, G.M., Ferreira, D.R.: Understanding Spaghetti Models with Sequence Clustering for ProM. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) BPM 2009. LNBIP, vol. 43, pp. 92–103, Springer, Heidelberg (2010).
 18. WSMML 2 Reasoner Framework (WSMML2Reasoner). <http://tools.deri.org/>

From Declarative Processes to Imperative Models

Johannes Prescher, Claudio Di Ciccio, and Jan Mendling*

Vienna University of Economics and Business, Vienna, Austria

{johannes.prescher, claudio.di.ciccio, jan.mendling}@wu.ac.at

Abstract. Nowadays organizations support their creation of value by explicitly defining the processes to be carried out. Processes are specifically discussed from the angle of simplicity, i.e., how compact and easy to understand they can be represented. In most cases, organizations rely on imperative models which, however, become complex and cluttered when it comes to flexibility and optionality. As an alternative, declarative modeling reveals to be effective under such circumstances. While both approaches are well known for themselves, there is still not a deep understanding of their semantic interoperability. With this work, we examine the latter and show how to obtain an imperative model out of a set of declarative constraints. To this aim, we devise an approach leading from a Declare model to a behaviorally equivalent Petri net. Furthermore, we demonstrate that any declarative control flow can be represented by means of a Petri net for which the property of safety always holds true.

1 Introduction

The definition of valid behavior is at the core of every organization in order to support the creation of value. Such behavior is in most cases modeled using an imperative concept, e.g., by means of notations such as Petri nets [1] or BPMN [18]. They explicitly describe the options to continue at each state. However, while imperative approaches are a strong concept when it comes to well-defined processes, they lack clarity once an observed behavior allows for flexible execution. In this case, models following a declarative approach are able to describe the behavior in a more compact way [4].

Recent research, however, acknowledges that hardly any of the available representations would be superior in all circumstances. For instance, it was pointed out that imperative and declarative models are favoring different types of comprehension tasks [19, 31]. Therefore, approaches have been proposed to represent a mined process partly as an imperative model and partly as a declarative model [35]. A problem in this context is, however, to choose which parts would better be shown in either way. In order to allow for an informed decision, a preliminary question has to be answered: is there a possibility to represent the same behavior regardless of the notation?

In this paper, we start answering this research question by describing how to derive an imperative model from a declarative one. We build upon existing work on transformations from Transition Systems to Petri nets by extending the approach to a tool chain

* The research leading to these results has received funding from EU Seventh Framework Programme (FP7) under grant agreement 318275 (GET Service).

that leads from a Declare model to a behaviorally equivalent Petri net. We implemented and tested our approach using the logs of the BPI Challenge from 2013. Lastly, we show that the imperative version always holds the property of safety.

The paper is structured as follows. Section 2 defines the background of our research, namely preliminaries of different representations including automata, transition systems, Petri nets, and Declare. Section 3 defines our transformation approach. Section 4 demonstrates the feasibility of our approach using a prototypical implementation applied to the BPI Challenge 2013. Section 5 discusses related work before Section 6 concludes the paper.

2 Background

In this section, we discuss Finite State Automata as generic, yet verbose representations of behavior. Then, we revisit the essential concepts of Petri nets. Finally, we introduce Declare as a representation based on behavioral constraints.

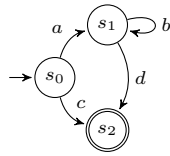


Fig. 1: A process behavior as an FSA

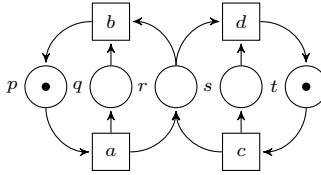


Fig. 2: A Petri net \mathcal{P}

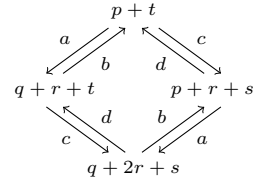


Fig. 3: The Reachability Graph of \mathcal{P}

2.1 Finite State Automata

In general, a process can be described as a *stateful* artifact characterized by its conversational behavior, i.e., its potential evolutions resulting from the interaction with some external system, such as a client service. The finite set of possible interactions constitutes the so-called *process alphabet*. The conversational behavior can be represented as a *Finite State Automaton (FSA)*. Its transitions are labeled by process activities, under the assumption that each legal run of the system corresponds to a conversation supported by the process. A process behavior is represented by a finite deterministic Transition System $\mathcal{S} = \langle \mathcal{A}, S, s_0, \delta, S^f \rangle$, where: \mathcal{A} is the process alphabet; S is the finite non-empty set of states; $s_0 \in S$ is the initial state; $\delta : S \times \mathcal{A} \rightarrow S$ is the transition function (by $s \xrightarrow{a} s'$ we denote that, from state s , transition a leads to state s'); $S^f \subseteq S$ is the set of final states.

The initial and final states respectively correspond to a legal initialization and termination of the process lifecycle. W.l.o.g., we assume that every state is reachable by traversing the automaton, starting from the initial state. Thus, in Figure 1, the process would admit the instance to either (i) perform activity a and then b an arbitrary number of times, and finally d , then terminate, or (ii) perform c once and terminate. We can consider FSAs to be for process modeling what Assembly is for computer programming. FSAs are simple and valuable in terms of expressive power, but have problems

modeling concurrency succinctly. Suppose that there are n parallel activities, i.e., all n activities need to be executed but any order is allowed. There are $n!$ possible execution sequences. The FSA thus requires $2n$ states and $n \times 2n - 1$ transitions. This is an example of the well-known “state explosion” problem. Concurrency is known to be well handled by Petri nets.

2.2 Petri nets

Petri nets (PNs) originate from the Ph.D. thesis of Carl Adam Petri [30]. A PN is a directed bipartite graph. Its vertices can be divided into two disjoint finite sets consisting of *places* and *transitions*. Every arc of a PN connects a place to a transition or vice versa, but neither two places nor two transitions can be directly connected. Formally, a Petri net is a tuple $\mathcal{P} = \langle P, T, F \rangle$, where:

- P is a finite set of *places*;
- T is a finite set of *transitions*;
- $F \subseteq (P \times T) \cup (T \times P)$ is the *flow relation*.

Places in a PN may contain a discrete number of marks called *tokens*. Any distribution of tokens over the places represents a configuration of the net called *marking*. Formally, a marking of a PN is a multiset of its places, i.e., a mapping $\mathcal{M} : P \rightarrow \mathbb{N}$. We say the marking assigns a number of tokens (graphically represented as black dots) to each place; it represents a state of the system and can be regarded as a vector of non-negative integers of length P . We thus denote a marking M as a linear combination of places, where the linear factor corresponds to the number of tokens in the place. In the following, we will adopt either the vectorial or the polynomial notation (e.g., $M_0 = (1, 0, 0, 0, 1)$ for states p, q, r, s, t in Figure 2, which we also denote as $p + t$) to this extent. A transition $t \in T$ of a PN may *fire* whenever there are sufficient tokens at the start of *all* input arcs; when it fires, it consumes these tokens, and puts tokens at the end of *all* output arcs. Thus, t leads from a marking $M_1 \in \mathcal{M}$ to a marking $M_2 \in \mathcal{M}$ ($M_1 \xrightarrow{t} M_2$). In other words, M_2 is *reachable* from M_1 by means of t . Firings are atomic, i.e., single non-interruptible steps. PNs are always associated to an *initial* marking M_0 , denoting the initial status of the described system. The set of all markings reachable from M_0 is called its *reachability set*. A PN with initial marking M_0 is k -bounded iff for every reachable marking M , no place contains more than k tokens (k is the minimal number for which this holds). A 1-bounded net is called *safe*. Figure 2 depicts a 2-bounded PN. A *labeled* Petri net is a PN with labeling function $\lambda : T \rightarrow \mathcal{A}$, which puts into correspondence every transition of the net with a symbol (called label) from the alphabet \mathcal{A} . Henceforth, we will refer to labeled PNs simply as PNs, for the sake of conciseness.

Thus, modeling a process in terms of a Petri net is rather straightforward: (i) *activities* are modeled by *transitions*; (ii) *conditions* are modeled by *places*; (iii) *cases* are modeled by *tokens*. Figure 2 depicts the parallel evolution of two separate branches of the execution, one involving a loop of c ’s and d ’s, the other involving loops of a ’s and b ’s.



















	Constraint	Regular expression	Notation
Existence constraints	<i>Participation</i> (a)	$[\neg a]^* \cdot (a[\neg a]^* + [\neg a]^* \cdot a)$	
	<i>AtMostOne</i> (a)	$[\neg a]^* \cdot (a) ? [\neg a]^*$	
	<i>Init</i> (a)	$a \cdot ^*$	
	<i>End</i> (a)	$\cdot ^* a$	
Relation constraints	<i>RespondedExistence</i> (a, b)	$[\neg a]^* \cdot ((a \cdot ^* b \cdot ^*) \mid (b \cdot ^* a \cdot ^*)) \cdot [\neg a]^*$	
	<i>Response</i> (a, b)	$[\neg a]^* \cdot (a \cdot ^* b) \cdot [\neg a]^*$	
	<i>AlternateResponse</i> (a, b)	$[\neg a]^* \cdot (a[\neg a]^* \cdot b[\neg a]^*) \cdot [\neg a]^*$	
	<i>ChainResponse</i> (a, b)	$[\neg a]^* \cdot (ab[\neg a]^*) \cdot [\neg a]^*$	
	<i>Precedence</i> (a, b)	$[\neg b]^* \cdot (a \cdot ^* b) \cdot [\neg b]^*$	
	<i>AlternatePrecedence</i> (a, b)	$[\neg b]^* \cdot (a[\neg b]^* \cdot b[\neg b]^*) \cdot [\neg b]^*$	
	<i>ChainPrecedence</i> (a, b)	$[\neg b]^* \cdot (ab[\neg b]^*) \cdot [\neg b]^*$	
Mutual relation constraints	<i>CoExistence</i> (a, b)	$[\neg ab]^* \cdot ((a \cdot ^* b \cdot ^*) \mid (b \cdot ^* a \cdot ^*)) \cdot [\neg ab]^*$	
	<i>Succession</i> (a, b)	$[\neg ab]^* \cdot (a \cdot ^* b) \cdot [\neg ab]^*$	
	<i>AlternateSuccession</i> (a, b)	$[\neg ab]^* \cdot (a[\neg ab]^* \cdot b[\neg ab]^*) \cdot [\neg ab]^*$	
	<i>ChainSuccession</i> (a, b)	$[\neg ab]^* \cdot (ab[\neg ab]^*) \cdot [\neg ab]^*$	
Negative relation constraints	<i>NotChainSuccession</i> (a, b)	$[\neg a]^* \cdot (aa^*[\neg ab][\neg a]^* \cdot ([\neg a]^* \cdot a))$	
	<i>NotSuccession</i> (a, b)	$[\neg a]^* \cdot (a[\neg b]^*) \cdot [\neg ab]^*$	
	<i>NotCoExistence</i> (a, b)	$[\neg ab]^* \cdot ((a[\neg b]^*) \mid (b[\neg a]^*)) ?$	

Table 1: Semantics of Declare constraints as POSIX Regular Expressions [17]

Reachability Graph and Bisimulation The *Reachability Graph* (*RG*) of a PN is a Transition System in which (i) the set of states is the reachability set (every state is thus a reachable marking), (ii) the alphabet coincides with the one of the net, and (iii) $M_1 \xrightarrow{t} M_2$ iff there exists a transition t in the net that leads from marking M_1 to M_2 . Figure 3 depicts the Reachability Graph for the PN of Figure 2. With a slight abuse of terminology, we will thus refer to the *bisimulation* [27] of a Petri net and a Transition System, meaning that the Reachability Graph of the PN and the Transition System (TS) are bisimilar. We recall here that bisimulation relation is a behavioral equivalence relation, which entails the impossibility for an external user to distinguish the behavior of the two systems. As a consequence, the two systems are trace-equivalent (see [22]).

2.3 Declare Constraints

The need for flexibility in the definition of some types of process has lead to an alternative to the classical “imperative” approach: the “declarative” one. The classical approach is called “imperative” (or also “procedural”) because it explicitly represents every step allowed by the process model at hand, by means of transitions (the possible actions to do) among places/states (the legal situations where the process can wait or terminate). This leads to the likely increase of graphical objects as the process allows more alternative executions. The size of the model, though, has undesirable effects on understandability and likelihood of errors – see for instance work on process modeling

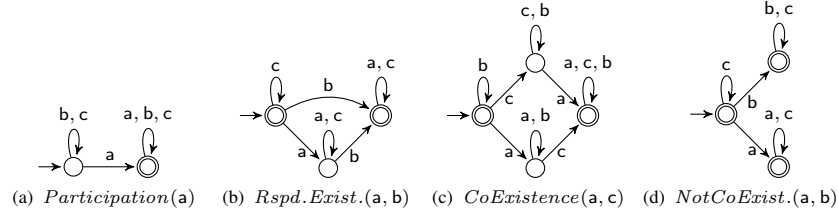


Fig. 4: FSAs accepting Declare constraints, on process alphabet $\mathcal{A} = \{a, b, c\}$

guidelines [25]. In fact, larger models tend to be more difficult to understand [26], not to mention the higher error probability from which they suffer, with respect to small models [24]. Rather than using a procedural language for expressing the allowed sequences of activities, it is based on the description of workflows through the usage of constraints: the idea is that every task can be performed, as long as its execution does not violate any of the specified constraints [29]. Declare [4] is a language defining an extensible set of *templates* for constraints. Declare constraint templates can be divided into two main types: *existence constraints* \mathcal{C}_E , and *relation constraints* \mathcal{C}_R . The former consists of constraint templates constraining single activities. As such, existence constraints can be expressed as predicates over one variable (the constrained activity): $\mathcal{C}_E(x)$. The latter comprises rules that are imposed on target activities, when activation tasks occur. Relation constraints thus correspond to predicates of arity two: $\mathcal{C}_R(x, y)$. Process alphabet \mathcal{A} is the domain of interpretation for constraints. Given a (possibly empty) set of existence constraints of size $m \geq 0$ (resp. relation constraints, of size $n \geq 0$) interpreted over alphabet \mathcal{A} , each denoted as $\mathcal{C}_{E_i}^{\mathcal{A}}(x)$ (resp. $\mathcal{C}_{R_j}^{\mathcal{A}}(x, y)$), the declarative model consists of their conjunction: $\mathcal{C}_{E_1}^{\mathcal{A}}(x) \wedge \dots \wedge \mathcal{C}_{E_m}^{\mathcal{A}}(x) \wedge \mathcal{C}_{R_1}^{\mathcal{A}}(x, y) \wedge \dots \wedge \mathcal{C}_{R_n}^{\mathcal{A}}(x, y)$.

Participation(a) is an existence constraint, which requires the execution of a at least once in every process instance. *AtMostOne*(a) is its dual, as it specifies that a is not executed more than once in a process instance. *End*(a) requires that a occurs in every case as the last activity carried out. *RespondedExistence*(a, b) is a relation constraint. It imposes that if a is performed at least once during the enactment of the process, b must be executed at least once as well, either in the future or in the past, with respect to the time in which a is carried out. *Response*(a, b) enforces *RespondedExistence*(a, b) by specifying that b must occur eventually *afterwards*. *AlternateResponse*(a, b) adds to *Response*(a, b) the condition that no other a's occur between an execution of a and a subsequent b. Two specializations of the relation constraints are *mutual relation constraints* and *negative relation constraints*. Mutual relation constraints are such that both constrained activities are activation and target. For instance, *CoExistence*(a, c) is a mutual relation constraint requiring that if a is executed, then c must be performed as well, and vice versa, in any order. Negative relation constraints are such that both constrained activities are activation and target as well. However, the occurrence of one activity *excludes* the occurrence of the other. For instance, *NotCoExistence*(a, b) is a negative relation constraint imposing that if a is executed, then b cannot be performed at all in the trace, and vice versa.

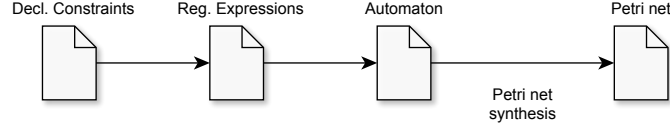


Fig. 5: Obtaining imperative processes as Petri nets from declarative constraints.

$NotSuccession(a, b)$ is a looser constraint, because it requires that no b 's occur after a (and therefore no a 's before b). $NotChainSuccession(a, b)$ requires that the next activity after a cannot be b . An example of graphical representation for a simple Declare process model is drawn in Figure 6a.

The semantics of Declare templates have been expressed as formulations of several formal languages: as Linear Temporal Logic over Finite Traces (LTL_f) formulas [14], in [13]; as *SCIFF* integrity constraints [6], in [8]; as First Order Logic (FOL) formulas, interpreted on finite traces, in [16], based on [14]; as Regular Expressions (REs) in [17]. In particular, our work will build upon the last translation, as explained in the next section. Table 1 reports the semantics of Declare constraints as REs. In the table, as well as in the remainder of this paper, we adopt POSIX standard shortcuts for REs, for the sake of brevity. Therefore, in addition to the known Kleene star ($*$), alternation ($|$) and concatenation ($()$) operators, we make use here of (i) the $.$ and $[\wedge x]$ shortcuts for respectively matching any character in the alphabet, or any character but x , (ii) the $+$ and $?$ operators for respectively matching from one to any, or none to one, occurrences of the preceding expression. We will also utilize the intersection operator $\&$ for REs.

3 Conceptual Framework

In this section, we show an approach that describes how to compute a Petri net corresponding to a Declare process model. This approach serves as a conceptual framework for proving that there always exists a Petri net which is *bisimilar* to a Declare process model. Furthermore, the returned PN is proven to be *safe*. The computation consists of three main steps, as sketched in Figure 5.

Declarative constraints to Regular Expressions. We represent all declarative constraints as REs. Each constraint maps to a single RE, i.e., the mapping is one-to-one (cf. Table 1). REs apply to characters. Owing to this, our approach identifies each activity in the process alphabet with a character.

Regular Expressions to Finite State Automaton. The allowed behavior is given by the conjunction of all Declare constraints. Hence, it maps to the *intersection* of the languages accepted by corresponding REs, i.e., the language accepted by the conjunction of the REs (which is in turn a RE itself, being Regular Expressions close w.r.t. the conjunction operation [21]). For the sake of conciseness, though, single REs are thought to directly refer to those activities (characters). They are constrained by the corresponding

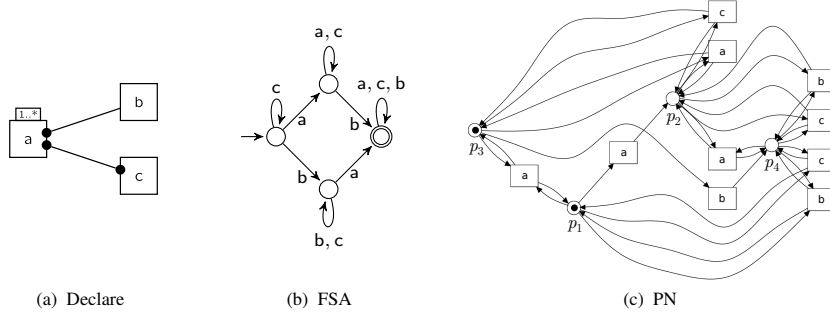


Fig. 6: A Declare model, consisting of constraints $Participation(a)$, $RespondedExistence(a, b)$ and $CoExistence(a, c)$, represented with the Declare graphical notation, as a Finite State Automaton and as a Petri net

constraint, disregarding the rest of the process alphabet in their formulation. Consider, for example, $Participation(a)$, depicted in Table 1. The corresponding RE requires the occurrence of a at least once, but also allows any other input beforehand and afterwards. Therefore, we need to limit the set of allowed characters to those which identify activities in the process alphabet \mathcal{A} (see Section 2). This is obtained by means of another RE, which is put in conjunction with the constraint-related ones. This way, we can define the declarative process model described by N constraints by means of a Regular Expression, derived from the conjunction of $N + 1$ REs.

As an example, we consider a process consisting of the following three constraints and having process alphabet $\mathcal{A} = \{a, b, c\}$:

- $Participation(a)$, translating to $[\hat{a}]^* (a[\hat{a}]^*) + [\hat{a}]^*$, referred to as (re1),
- $RespondedExistence(a, b)$, translating to $[\hat{a}]^* ((a.*b.*) | (b.*a.*)) * [\hat{a}]^*$, referred to as (re2), and
- $CoExistence(a, c)$, translating to $[\hat{a}\hat{a}\hat{c}]^* ((a.*c.*) | (c.*a.*)) * [\hat{a}\hat{a}\hat{c}]^*$, referred to as (re3).

The mere conjunction of (re1), (re2) and (re3) would still allow not only for the characters representing activities but also for any input character. In order to limit input characters to those which identify activities in the process alphabet, we thus conjunct the aforementioned three to the following: $([abc]^*)$. As a result, the final RE is: (re1) & (re2) & (re3) & $([abc]^*)$.

Continuing with our computation, we transform the RE into the corresponding FSA. We recall here indeed that regular grammars are recognizable through Regular Expressions [9]. Figures 4a to 4c depict the FSAs accepting the languages of (re1), (re2) and (re3), respectively. Figure 6b shows the FSA which results from the example we provided. Aside of the transitions that do not change its state, the FSA allows two different runs before reaching its final state, i.e. either $\langle a, b \rangle$ or $\langle b, a \rangle$.

Finite State Automaton to Petri net. In the last step of our approach, we derive a Petri net from the FSA. For this purpose, we rely on the theory of regions described in [12], adopted to synthesize PNs from state-based models, such as Transition Systems (and thus, *a fortiori*, FSAs). The rationale behind the theory of regions is to conglomerate sets of places that share the same input and output transitions in common *regions*. The regions translate to places in the derived PN. Input transitions lead to them, and output transitions start from them. In particular, we adopt the approach described in [11], which is proven to return a *safe* Petri net from a Transition System, ensuring the bisimilarity between the two systems (see Section 2.2).

Figure 6c shows the Petri net stemming from the application of the technique of Cortadella et al. [11] to the FSA of our example. Just as the FSA, it contains four places (p_1, p_2, p_3, p_4) and has the initial marking $M = (1, 0, 1, 0)$, i.e., it contains a token in p_1 and p_3 . However, the places of the Petri net do not correspond directly to the states of the FSA. Instead (again, without considering the firings that do not change the marking), just as the FSA, the PN allows two different runs ($\langle a, b \rangle$ and $\langle b, a \rangle$). As the final state of the FSA allows for the execution of any activity in the process alphabet (any character of the input alphabet), the PN also allows for this behavior when its marking is $M = (0, 1, 0, 1)$. Please note that such marking is reachable only by means of the sequence of firings that replicate the sequence of characters leading to the accepting state of the FSA.

The reader can notice that the returned net presents multiple transitions labeled the same, i.e., representing the same activity. This is due to the fact that label-splitting can be avoided for derived safe PNs only if the Transition System has the property of excitation closure for its transitions, i.e., only if the intersection of those states from which the transitions start can be grouped in one single activating region [11]. However, such property is not guaranteed from the FSAs that Declare processes translate to. Later work of Carmona et al. [7] shows how to balance the trade-off between k -boundedness of the returned Petri net and the number of splitted labels.

To sum up, applying the steps mentioned above, we derive an imperative model from declarative constraints. Note that the operations we perform are transformations that do not alter the behavior. Thus, not only the declarative constraints but also the Regular Expression, the FSA and the PN represent the same behavioral characteristics of the process. Furthermore, we have demonstrated by construction the following theorem.

Theorem 1. *Given any Declare process model \mathcal{P}_D consisting of $n \geq 0$ existence constraints and $m \geq 0$ relation constraints, expressed over process alphabet \mathcal{A} , $\mathcal{P}_D = \bigwedge_{i=1}^m \mathcal{C}_{E_i}^A(x) \wedge \bigwedge_{j=1}^n \mathcal{C}_{R_j}^A(x, y)$, there always exist a safe Petri net model $\mathcal{P}_N = \langle P, T, F \rangle$ labeled by $\lambda : T \rightarrow \mathcal{A}$, which is bisimilar to \mathcal{P}_D and is safe.*

4 Evaluation by Implementation

In this section, we present a feasibility evaluation of our proposed concepts based on a prototypical implementation. We first describe the implementation. Then, we present the results of its application on a Declare model generated from a log of the BPI Challenge 2013. Finally, we discuss insights from the case.

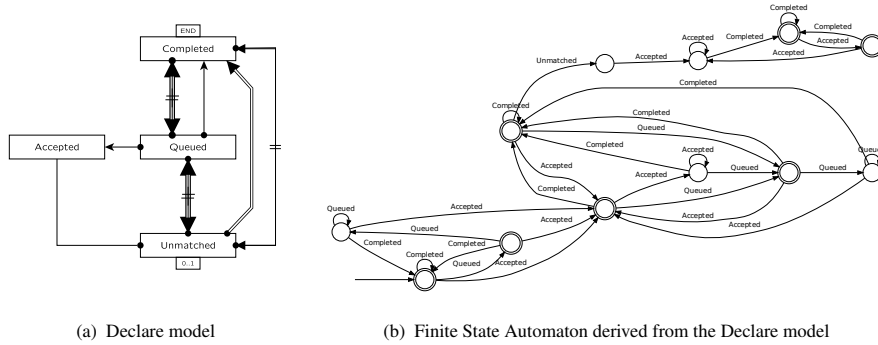


Fig. 7: The process mined out of BPIC 2013 log [33], as Declare model and FSA

4.1 Implementation

In order to have the opportunity to analyze real-life declarative process models, we have integrated our approach with a tool for the mining of declarative control flows from event logs (see Figure 5), namely MINERful [15]. The MINERful framework comes with an integrated support of a library called *dk.bricks.automaton* [28], for the generation of FSAs out of Regular Expressions. We extended the integrated MINERful-*dk.bricks.automaton* tool in order to make it capable of serializing FSAs into TSML-encoded files. TSML (Transition System Markup Language) is indeed a format supported by ProM, the Process Mining Toolkit [3]. In this way, we have been able to apply the ProM plug-in by van Dongen (see [5]), capable of converting a TSML-encoded Transition System into a Petri net, by using Petrify (see [10]).

4.2 Application to the BPI Challenge 2013

As a real-world data set for validating the approach, we selected the “BPI Challenge 2013, closed problems” log [33] as an application case. For the control-flow discovery task, we have considered the activities’ names as their identifiers (Accepted, Completed, Queued and Unmatched). We have set MINERful up in order to return those constraints proven to be valid in every trace (support threshold equal to 1). The discovered model consisted of the following 10 constraints:

<i>Response</i> (Queued, Accepted)	<i>End</i> (Completed)
<i>NotChainSuccession</i> (Queued, Completed)	<i>NotSuccession</i> (Completed, Unmatched)
<i>Response</i> (Queued, Completed)	<i>AtMostOne</i> (Unmatched)
<i>NotChainSuccession</i> (Queued, Unmatched)	<i>RespondedExistence</i> (Unmatched, Accepted)
<i>Response</i> (Accepted, Completed)	<i>AlternateResponse</i> (Unmatched, Completed)

The graphical representation of the model is depicted in Figure 7a. Figure 7b draws the Finite State Automaton derived from the Declare model, and Figure 8 shows the final outcome, as a Petri net. What we can observe from the comparison of the Declare model and the behavior-equivalent Petri net is the multiplication of various activities.

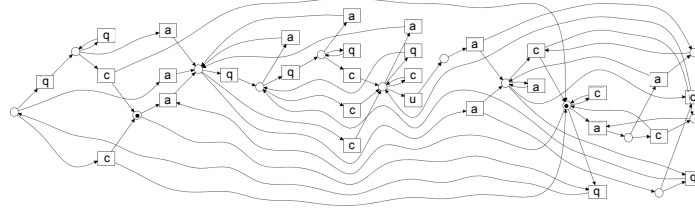


Fig. 8: The Petri net derived from the Finite State Automaton representing the Declare model mined out of BPIC 2013 log [33]. Labels are abbreviated: a = “Accepted”, c = “Completed”, u = “Unmatched”, q = “Queued”.

Although the Declare model seems to be more compact in terms of its nodes and edges, it must be noted that the Petri net is presented as it was produced, i.e., it has not been subject to any post-processing for reducing its complexity. However, the defined chain of transformations provide us with the basis to study the trade-off between compactness of the model and richness of the language in future experiments, conducted on behavior-equivalent Petri nets and Declare models.

5 Related work

The stream of research on the comparison of declarative and imperative modeling approaches has been discussed considering different perspectives. In [32], Pichler et al. investigate both imperative and declarative languages with respect to process model understanding. The issue of maintainability for both languages is discussed in [20]. An open problem for this experimental stream of research has been the question of what a fair comparison is for declarative and imperative models. In this regard, the work of [35] and [36] elaborates on mixed representations as a combination of both approaches from a modeling perspective.

Furthermore, research on automatic process discovery techniques has been defined based on different representations and different techniques of discovery beyond the classical alpha miner [2]. Our selection is not meant to be exhaustive, but rather highlights those approaches that use constraints, automata or transition systems. Van der Aalst et al. propose a two-step approach in [5] in order to discover transition systems which are then synthesized to Petri nets using the “theory of regions”. As well as Van der Aalst et al., Maruster et al. suggested an approach for process discovery in which they deal with noise and imbalance in process logs ([23]). A tool manipulating concurrent specifications, synthesis and optimization of asynchronous controllers is presented in [10]. In order to come up with a better understanding of the mutual strengths and weaknesses of these approaches, De Weerd et al. ([34]) provide an extensive, multi-dimensional survey of existing process discovery algorithms using real-life event logs. Different representations are, however, not discussed in this survey. In this way, our work provides a basis for an extensive comparison in the future.

6 Conclusion

In this paper, we described an approach to derive imperative process models from declarative process control-flows. To this extent, we utilize a sequence of steps, leading from declarative constraints to Regular Expressions, then to a Finite State Automaton, and finally to a Petri net. We implemented our integrative approach as part of the MINERful software package and evaluated it using the real world case of the BPI Challenge 2013. A remaining limitation is that we do not provide a sound solution for a transformation from an arbitrary imperative model into a declarative representation. In future research, we will address this issue. Furthermore, we plan to utilize the transformation in the design of experiments to study the mutual benefits of PNs and Declare models in model comprehension tasks.

References

1. van der Aalst, W.M.P.: The application of petri nets to workflow management. *Journal of Circuits, Systems, and Computers* 8(1), 21–66 (1998)
2. van der Aalst, W.M.P.: *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer (2011)
3. van der Aalst, W.M.P., van Dongen, B.F., Günther, C.W., Rozinat, A., Verbeek, E., Weijters, T.: ProM: The process mining toolkit. In: *BPM (Demos)* (2009)
4. van der Aalst, W.M.P., Pesic, M.: DecSerFlow: Towards a truly declarative service flow language. In: *WS-FM*. pp. 1–23 (2006)
5. van der Aalst, W.M.P., Rubin, V., Verbeek, H.M.W., van Dongen, B.F., Kindler, E., Günther, C.W.: Process mining: a two-step approach to balance between underfitting and overfitting. *Software and System Modeling* 9(1), 87–111 (2010)
6. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Verifiable agent interaction in abductive logic programming: The sciff framework. *ACM Trans. Comput. Log.* 9(4), 29:1–29:43 (2008)
7. Carmona, J., Cortadella, J., Kishinevsky, M.: New region-based algorithms for deriving bounded petri nets. *IEEE Trans. Computers* 59(3), 371–384 (2010)
8. Chesani, F., Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Exploiting inductive logic programming techniques for declarative process mining. *T. Petri Nets and Other Models of Concurrency* 2, 278–295 (2009)
9. Chomsky, N., Miller, G.A.: Finite state languages. *Information and Control* 1(2), 91–112 (1958)
10. Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions* 80(3), 315–325 (1997)
11. Cortadella, J., Kishinevsky, M., Lavagno, L., Yakovlev, A.: Deriving petri nets from finite transition systems. *IEEE Trans. Comput.* 47(8), 859–882 (1998)
12. Cortadella, J., Kishinevsky, M., Lavagno, L., Yakovlev, A.: Synthesizing petri nets from state-based models. In: *Proc. of ICCAD’95*. pp. 164–171 (November 1995)
13. De Giacomo, G., De Masellis, R., Montali, M.: Reasoning on ltl on finite traces: Insensitivity to infiniteness. In: *AAAI* (2014)
14. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: *IJCAI* (2013)
15. Di Ciccio, C., Mecella, M.: A two-step fast algorithm for the automated discovery of declarative workflows. In: *Proc. of CIDM, Singapore 2013*. pp. 135–142. IEEE (2013)

16. Di Ciccio, C., Mecella, M.: On the discovery of declarative control flows for artful processes. *ACM Transactions on Management Information Systems* (2014)
17. Di Ciccio, C., Mecella, M., Scannapieco, M., Zardetto, D., Catarci, T.: MailOfMine – analyzing mail messages for mining artful collaborative processes. In: *Data-Driven Process Discovery and Analysis*, vol. 116, pp. 55–81. Springer (2012)
18. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: *Fundamentals of Business Process Management*. Springer (2013)
19. Fahland, D., Lübke, D., Mendling, J., Reijers, H.A., Weber, B., Weidlich, M., Zugal, S.: Declarative versus imperative process modeling languages: The issue of understandability. In: *BMMDS/EMMSAD*. pp. 353–366 (2009)
20. Fahland, D., Mendling, J., Reijers, H.A., Weber, B., Weidlich, M., Zugal, S.: Declarative versus imperative process modeling languages: The issue of maintainability. In: *Business Process Management Workshops*. pp. 477–488 (2009)
21. Gisburg, S., Rose, G.F.: Preservation of languages by transducers. *Information and Control* 9(2), 153 – 176 (1966)
22. van Glabbeek, R.J., Weijland, W.P.: Branching time and abstraction in bisimulation semantics. *J. ACM* 43(3), 555–600 (1996)
23. Maruster, L., Weijters, A.J.M.M., van der Aalst, W.M.P., van den Bosch, A.: A rule-based approach for process discovery: Dealing with noise and imbalance in process logs. *Data Min. Knowl. Discov.* 13(1), 67–87 (2006)
24. Mendling, J., Neumann, G., van der Aalst, W.M.P.: Understanding the occurrence of errors in process models based on metrics. In: *OTM Conferences* (1). pp. 113–130 (2007)
25. Mendling, J., Reijers, H.A., van der Aalst, W.M.P.: Seven process modeling guidelines (7PMG). *Information & Software Technology* 52(2), 127–136 (2010)
26. Mendling, J., Reijers, H.A., Cardoso, J.: What makes process models understandable? In: *BPM*. pp. 48–63 (2007)
27. Milner, R.: An algebraic definition of simulation between programs. In: *IJCAI*. pp. 481–489 (1971)
28. Møller, A.: *dk.bricks.automaton* (2011)
29. Pesic, M., Schonenberg, M.H., Sidorova, N., van der Aalst, W.M.P.: Constraint-based workflow models: Change made easy. In: *OTM Conferences* (1). pp. 77–94 (2007)
30. Petri, C.A.: *Kommunikation mit Automaten*. Ph.D. thesis, Institut für instrumentelle Mathematik, Bonn (1962)
31. Pichler, P., Weber, B., Zugal, S., Pinggera, J., Mendling, J., Reijers, H.A.: Imperative versus declarative process modeling languages: An empirical investigation. In: *Business Process Management Workshops* (1). pp. 383–394 (2011)
32. Pichler, P., Weber, B., Zugal, S., Pinggera, J., Mendling, J., Reijers, H.A.: Imperative versus declarative process modeling languages: An empirical investigation. In: *Business Process Management Workshops* (1). pp. 383–394 (2011)
33. Steeman, W.: Real-life event logs – an incident management process: closed problems. *Third International Business Process Intelligence Challenge (BPIC’13)* (2013)
34. Weerdt, J.D., Backer, M.D., Vanthienen, J., Baesens, B.: A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs. *Inf. Syst.* 37(7), 654–676 (2012)
35. Westergaard, M., Slaats, T.: Cpn tools 4: A process modeling tool combining declarative and imperative paradigms. In: *BPM (Demos)* (2013)
36. Westergaard, M., Slaats, T.: Mixing paradigms for more comprehensible models. In: *BPM*. pp. 283–290 (2013)

Generating Artificial Event Logs with Sufficient Discriminatory Power to Compare Process Discovery Techniques

Toon Jouck¹ and Benoît Depaire^{1,2}

¹ Hasselt University, Faculty of Business Economics
Agoralaan Bldg D, 3590 Diepenbeek, Belgium
toon.jouck@uhasselt.be; benoit.depaire@uhasselt.be

² Research Foundation - Flanders (FWO)
Egmontstraat 5, 1000 Brussels, Belgium

Abstract. Past research revealed issues with artificial event data used for comparative analysis of process mining algorithms. The aim of this research is to design, implement and validate a framework for producing artificial event logs which should increase discriminatory power of artificial event logs when evaluating process discovery techniques.

Key words: Artificial Event Logs; Event Log Simulation; Performance Measurement of Business Processes

1 Research Question

Literature on the comparative analysis of process discovery techniques has revealed some problems with artificial data. The data lacked discriminatory power. We argue that such problems arose due to the absence of a proper framework to generate artificial data. This leads to our main research question: how can we generate artificial event logs with sufficient discriminatory power for a comparative evaluation of process discovery algorithms? To provide an answer to this question several other questions need to be answered:

- What model characteristics can we identify which influence the generated data?
- What is the impact of model language bias on the generated data?
- Which non-model characteristics exist which influence the generated data?
- What is a proper methodology for generating artificial data for comparative analysis?
- Which tools exist for generating artificial data and to what extent are they sufficient?

2 Background

This work focusses on artificial data used for the comparison of different process discovery techniques, more specifically the comparison of control-flow techniques.

In past research on process mining many researchers used artificial data for the development of and the verification of new algorithms (e.g. [1, 2]).

In a recent study De Weerd et al. compare several process discovery techniques on both artificial and real data [3]. The artificial data used in their experiments was recovered from past research on the development of a process discovery algorithm [2]. Remarkably, the performance of the algorithms did not seem to be significantly different for the artificial data, while real data revealed significant performance differences. These results indicate that the artificial test data used in past research have insufficient discriminatory power.

A lot of process discovery techniques have been developed in the last decade. Since the first algorithms, process discovery has matured remarkably. However, it's still not clear which algorithm will perform best in a certain situation. This has led to an increasing importance of the research on comparing different process discovery techniques [3, 4, 5].

3 Significance

The comparison of process discovery techniques can be based on both artificial and real data. Real data, however, are at a disadvantage when performing such a comparative analysis.

Two disadvantages stem from the nature of algorithm comparison and evaluation. Typically research is performed on a sample of event logs, but conclusions are preferably generalizable to other event logs. To achieve reliable conclusions, statistics require sufficient observations and samples which are representative for the considered population. Real data, however, have limited availability and are typically convenience samples, rather than random samples.

Another disadvantage of real data is concerned with identifying causal relationships between process or event log characteristics on the one hand and algorithms performance on the other. This kind of research requires experimental data and not observational data (real data).

In contrast, these disadvantages are not present when using artificial data in comparative analysis of process discovery techniques if a proper methodology is used to generate the artificial data. Such a methodology should focus on creating artificial data with sufficient discriminatory power to overcome the problems encountered in past research (e.g. [3]). The main contribution of this research will be drawing up and implementing a general methodology for the generation of artificial event logs with sufficient discriminatory power in order to evaluate process mining algorithms.

4 Research design and methods

Firstly, a structured literature review is performed to get insight into generating artificial data and algorithm comparison. The primary sources used to perform

this review are: literature in the domain of process mining and literature from other domains on (generating) synthetic data.

Secondly, the general methodology is built and implemented in a tool to support this new methodology.

Finally the implemented methodology is tested and validated by repeating experiments done in past research on comparing process discovery techniques.

One important limitation of this methodology will be its scope which is limited to generating artificial data for analysing control-flow discovery techniques. Also the reader should be aware that such a general methodology for artificial data will not replace the need for real (test) data. Real logs continue to be necessary for making artificial event logs more realistic and as a final review for process discovery techniques.

5 Research stage

5.1 A Preliminary Framework

The literature review of articles on the evaluation of process discovery techniques based on artificial data (i.a. [1, 2, 3]) reveals that there are only some guidelines or recurring elements for generating artificial logs. However, a sound and general methodology is missing, which decreases the relevance of artificial logs.

To address this issue a preliminary framework is distilled from the literature review which focusses on the crucial aspect of randomization. The methodology can be divided into two stages: the generation of an artificial process model and the generation of event logs from this model. Both stages allow the researcher to define the characteristics of the population and produce a representative sample (see table 1).

The first step is to define a population of process models, from which artificial models are sampled randomly and automatically. In past research this crucial step in generating artificial event logs was never made explicit in a general method or guideline. Mostly processes were drawn manually in an ad-hoc manner without explicitly defining the population they were drawn from. However, it is important that the researcher has insight into the process model population and can influence the properties of that population. Therefore, ranges for the list of controllable properties (see step 1 in table 1) must be set to define the population. Next, values within these ranges are selected randomly and automatically to define a single process model.

The second step concerns the generation of event logs for each process model defined in the previous stage. Again, the researcher must set ranges for several event log properties, from which exact values are sampled randomly to generate event logs. The parameters which can be set are shown in step 2 in table 1.

5.2 Tools for Generating Artificial Event Logs

Different tools already exist which can help to automatically generate artificial event logs. We evaluated two tools considered most appropriate to support the

Table 1. Preliminary Methodology for Generating Artificial Event Logs

Step Methodology	Controllable Properties
1. Model Generation	Number of activity types Choice structural patterns Choice nested structural patterns
2. Log Generation	Number of generated process instances Required completeness Noise Imbalance of execution properties

preliminary framework: the PLG tool [6] and the BeehiveZ tool [7]. At first sight both tools seem appropriate because both support the two stages of the proposed methodology. However, a more detailed evaluation revealed that both tools do not completely support the proposed methodology and several limitations exist. The results of the evaluation, summarized in table 2, show that the PLG tool supports the preliminary framework the best.

Table 2. Tools for Generating Artificial Event Logs

Properties	PLG	BeehiveZ
Number of activity types	NOK	OK
Choice structural patterns	OK	NOK (indirectly by generator)
Choice nested structural patterns	OK	NOK (indirectly by generator)
Number of generated process instances	OK	NOK
Required completeness	NOK	NOK (only for simple models)
Noise	OK	OK
Imbalance of execution properties	OK	NOK

5.3 A First Step Towards Validation

Although the framework as presented in table 1 is still preliminary, it was used in a first case study to assess if it was a step towards artificial data with more discriminatory power.

For this case study we repeat part of the experiment of De Weerd et al. [3] in which they evaluated process discovery techniques on both artificial and real event logs. Remarkably, the performance of the algorithms did not seem to be significantly different for the artificial data, while real event logs revealed significant performance differences.

We hypothesize that our methodology can produce artificial data with more discriminatory power. Therefore we repeat part of the experiment of De Weerd

et al. [3] on artificial data generated with the proposed methodology to see if our results are closer to the results on real data in De Weerd et al., than their own results on artificial data. If that is true, the case study will provide a first support to our hypothesis.

We applied the preliminary methodology to generate 35 artificial event logs out of two random populations using the PLG tool (with all its limitations). Then four process discovery algorithms were evaluated in two conformance dimensions, fitness and precision, using the method described in [3].

The results in the fitness dimension show that the performance of the tested algorithms reflect better the results for fitness on real data in [3], and thus supports the earlier stated hypothesis. However, the performance differences with respect to fitness in our experiments were of a different order of magnitude than the performance differences based on real data found by De Weerd et al. [3]. Moreover, the results from our experiments don't show any significant differences in terms of precision in contrast to the results in [3] based on real data.

From these findings can be concluded that the preliminary methodology is only a first step in the direction of increasing the discriminatory power of artificial event logs.

References

1. van der Aalst, W., Weijters, T., Maruster, L.: Workflow mining: discovering process models from event logs. *Knowledge and Data Engineering, IEEE Transactions on* **16**(9) (September 2004) 1128 – 1142
2. de Medeiros, A.K.A.: Genetic process mining. PhD thesis, Technische Universiteit Eindhoven (2006)
3. De Weerd, J., De Backer, M., Vanthienen, J., Baesens, B.: A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs. *Information Systems* **37**(7) (November 2012) 654–676
4. Rozinat, A., de Medeiros, A.A., Gnther, C.W., Weijters, A., van der Aalst, W.M.: Towards an evaluation framework for process mining algorithms. In: BPM Center Report, BPMcenter.org (2007)
5. vanden Broucke, S.K., Delvaux, C., Freitas, J., Rogova, T., Vanthienen, J., Baesens, B.: Uncovering the relationship between event log characteristics and process discovery techniques. In: Business Process Management Workshops, Springer (2014) 41–53
6. Burattin, A., Sperduti, Alessandro, A.: PLG: a process log generator. Technical report
7. Jin, T., Wang, J., Wen, L.: Efficiently querying business process models with BeehiveZ. In: BPM (Demos), Clermont-Ferrand, France (2011)

Process Mining Extension to SCAMPI

Arthur Valle^{1,2}, Eduardo Rocha Loures¹, Eduardo Portela¹

¹Pontifical Catholic University of Parana, Industrial and Systems Engineering, Curitiba, Brazil
{arthur.maria, eduardo.loures, eduardo.portela}@pucpr.br

²Wipro Technologies, Curitiba, Brazil
arthur.valle@wipro.com

Abstract. Existing process assessment methods, such as SCAMPI-Standard CMMI Appraisal Method for Process Improvement, do not use contemporary data collection and analysis techniques like processes mining, text mining or data mining. On the contrary, they use traditional ones: questionnaires, document review, interviews and demonstrations. Process mining is a technique that can be used to aid process assessments, aiming to conduct them with greater deepness and coverage, while keeping similar level of effort. The purpose of the PhD work is to develop a framework (structure and content) to apply process mining techniques in SCAMPI assessments.

1 Introduction

The present paper proposes the Process Mining Extension to SCAMPI, a framework where process mining techniques are added to existing assessment techniques. The paper is organized as follows. Section 2 presents the research question. Section 3 describes the background. Section 4 presents the significance of the work. Section 5 presents the research design and method. Section 6 provides the research stage.

2 Research Question

The research question can be stated as: “Comparatively with traditional SCAMPI assessments, does the proposed method extension enable software process assessments with more objectivity, accuracy, depth and coverage of aspects related to the execution of processes, while maintaining similar levels of effort, cost and time?”.

3 Background

SCAMPI [1] is a method used to assess organizations that use CMMI as a reference for their operations (software development, service management, etc). The fundamental idea behind the SCAMPI, as well as other similar assessments, is that the conduction of an activity or process results in "footprints" called objective evidences,

which are evaluated by experts to judge whether they satisfy best practices of a given CMMI reference model [2].

According to Fig. 1, extracted from Process Mining Manifesto, Process Mining is a set of techniques, tools, and methods to discover, monitor and improve real processes (i.e., not assumed processes) by extracting knowledge from event logs commonly available in today's (information) systems [3].

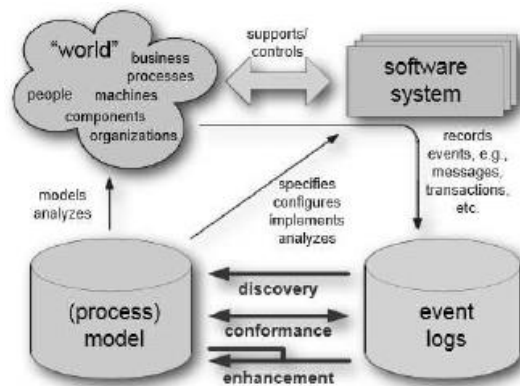


Fig. 1. Process Mining Overview, reproduced from [3]

There are three main types of process mining techniques [3]: a) Process Discovery (from an event log, a “as is” process model is identified); b) Conformance (or Compliance) Checking (an existing process model is compared to an event log of the same process); c) Enhancement (a process model is improved using information extracted from a log).

In order to identify work that proposed the use of process mining in process assessment, a systematic literature review was conducted. As a result of the application of a defined criterion and procedure, in six renowned scientific databases, only 6 out of 26 resulting papers were selected. Since none of them mentioned which process assessment methods were used, an additional search on Google Scholar was conducted using the same terms, resulting in some relevant papers, as follows:

- PhD thesis of Samalik, entitled Process Mining Application in Software Process Assessment [4]. The objective was to promote the use of process mining in software process assessment and improvement. Her conclusion was that techniques for collecting information derived from process mining can be applied to improve the data collection on software process assessment. However, conclusion was reached by qualitative judgment without objective criteria. In addition, process mining techniques that should be used were not nominally listed.
- Master dissertation of Cruañas, entitled Process Mining Opportunities for CMMI Assessments [5]. The objective was to investigate the literature concerning support tools to find out if it is possible to use process mining to improve the assessment of CMMI. His conclusion was that process mining can not only help improve the current CMMI assessments, but can also be a useful tool to assist data collection.

However, conclusions were based on the generalization of processes mining techniques and perspectives without using objective criteria. Moreover, no process mining technique in particular was pointed out.

- A third paper found is [6] where different aspects of processes mining were addressed, such as control perspective, information perspective and organizational perspective. Some algorithms such as alpha algorithm, heuristics miner, genetic miner, social network miner, organizational miner and activity miner, which can be applied, were cited in the paper.

Although these papers demonstrate application of process mining in process assessments, there is no formal guidance of how to conducting it, covering for instance, how to capture business rules, how to compare models and logs, which process mining algorithms to use, and when, etc.

4 Significance

According to [7], existing process assessment methods (such as SCAMPI) have limitations: they are manual, time-consuming, inefficient, subjective and generally require experienced appraisers. However, these days, detailed information about processes is recorded in the form of event logs, transaction logs, databases, etc. In this sense, in a process assessment is no longer justifiable that only a small set of processes are checked. Instead, the entire process and all its instances should be considered, as long as this represents low costs, naturally. Additionally, in existing assessment methods, the following techniques are used for gathering information about the running processes in an organization: questionnaires, document review, interviews and demonstrations. It means that no contemporary data collection and analysis techniques such as data mining, text mining or process mining is used.

Therefore, it is proposed the application of process mining techniques on the SCAMPI. It means that event logs of software processes would be used to understand past and current situation in a complete, economical, reliable and accurate manner, thereby contributing to the collection and analysis of data, which are critical activities in any software process assessment method.

The premise is that nowadays companies have been extremely efficient in collecting, organizing, and storing a large amount of data obtained in their daily operations. Most of these companies, however, do not use such data properly so as to transform them into knowledge to be employed in assessment activities. The need of companies to learn more about how their processes actually operate is a major driver behind the development and increasing use of process-mining techniques.

The main objective of this work is to develop a framework for the application of process mining techniques in SCAMPI-based assessments. This framework aims at enable software process assessments with more objectivity, accuracy, depth and coverage of aspects related to the execution of processes (such as duration and sequence of activities, start and end dates and records of who were the real executors), while maintaining similar levels of effort, cost and time.

5 Research design and methods

According to Fig 2, the proposed framework is a structure that serves as a guide for applying process mining techniques in SCAMPI assessments. The intention is that such guidance could be seen as an extension to SCAMPI method description. Analog approaches already exist such as the SAFE extension to CMMI-DEV [8]. It means that the Process Mining Extension to SCAMPI adds (or modifies) content to the current SCAMPI method. Its content covers the application of process mining aspects in SCAMPI method. Typically it comprises full or partial processes or activities, although any element can be added or expanded, such as inputs, outputs, tools and techniques.

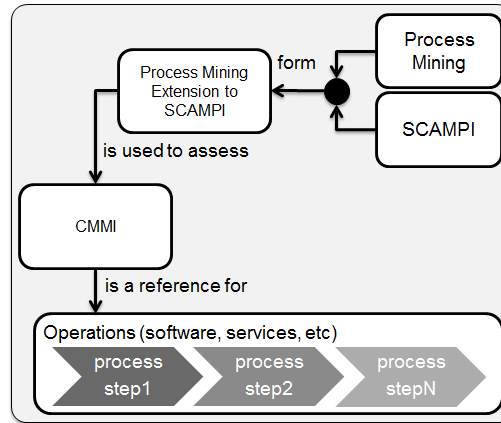


Fig. 2. Process Mining Extension to SCAMPI

Process Mining Extension to SCAMPI is a document with the following chapters: Executive Summary; Abstract; 1-Introduction; 1.1- Background and Acknowledgements; 1.2-Purpose and Scope; 1.3-Relationships with CMMI and SCAMPI; 1.4-Structure of the Process Mining Extension to SCAMPI; 1.5-Intended Audiences; 1.6-Usage scenarios; 2-Content; Appendix A: References; Appendix B: Acronyms; Appendix C: Glossary; Appendix D: Contact.

In order to define which content would be added or modified in Process Mining Extension to CMMI, some references were considered, beyond the SAFE extension to CMMI-DEV. For instance, [9] have proposed process mining use cases – typical applications of process mining functionalities in practical situations – to be used for detailed evaluation of process mining tools. Here, these use cases are used to identify typical process mining situations that are pertinent and could be applied in SCAMPI assessment. From the original list of 19 use cases, the following ones were taken in consideration: from Discovery perspective, UC1-Structure of the process; from Conformance Checking perspective, UC6-The degree in which the rules are obeyed and UC7-Compliance to the explicit model.

6 Research stage

The Process Mining Extension to SCAMPI is under development. Some chapters of the document are more advanced than others, such as 1.4-Structure of the Process Mining Extension to SCAMPI, 1.6-Usage scenarios and 2-Content.

In addition, the implementation of a running example to apply process mining techniques in SCAMPI assessments was conducted. The approach seemed to be feasible, as demonstrated using Disco and ProM process mining tools. Process mining techniques are demanded in order to transform existing process assessment methods, such as SCAMPI, into more productive and economically viable methods. Process mining enables an easy comparison on how processes are performed in practice versus the way they are designed to operate.

As future work, it is intended to use the framework in real SCAMPI assessments and to conduct statistical analysis and hypothesis thesis of performance parameters such as effort, duration, coverage and quality of results to quantitatively evaluate the benefits.

References

1. SCAMPI Upgrade Team.: Standard CMMI Appraisal Method for Process Improvement (SCAMPI), Version 1.3a: Method Definition Document for SCAMPI A, B and C (CMU/SEI-2013-HB-001). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University. <http://cmmiinstitute.com/resource/standard-cmmi-appraisal-method-process-improvement-scampi-b-c-version-1-3a-method-definition-document/>
2. Chrissis, M. B., Konrad, M., & Shrum, S.: CMMI for Development: Guidelines for Process Integration and Product Improvement. Addison-Wesley Professional (2011)
3. Van der Aalst, W. et al.: Process mining manifesto. In: Business process management workshops p. 169-194. Springer Berlin Heidelberg. (2012)
4. Samalik, J.: Process mining application in software process assessment. Eindhoven: Technische Universiteit Eindhoven. ((Co-)promot.: prof.dr. R.J. Kusters, prof.dr.ir. P.W.P.J. Grefen & dr.ir. J.J.M. Trienekens) (2012)
5. Cruaños, J. R.: Process Mining Opportunities for CMMI Assessments (2012)
6. Rubin, V. et al.: Process mining framework for software processes. In: Software Process Dynamics and Agility. Springer Berlin Heidelberg. p. 169-181 (2007)
7. Rout, T. P., El Emam, K., Fusani, M., Goldenson, D., & Jung, H. W.: SPICE in retrospect: Developing a standard for process assessment. Journal of Systems and Software, v. 80, n. 9, p. 1483-1493 (2007)
8. SAFE, V. 2: A Safety Extension to CMMI-DEV, V1. 2, Defence Materiel Organisation, Australian Department of Defence, Software Engineering Institute, TECHNICAL NOTE CMU. SEI-2007-TN-006 <http://www.sei.cmu.edu/pub/documents/07.reports/07tn006.pdf>. (2007)
9. Ailenei, I., Rozinat, A., Eckert, A. & van der Aalst, W.M.P.: Definition and Validation of Process Mining Use Cases. In F. Daniel, K. Barkaoui, and S. Dustdar, editors, Business Process Management Workshops, International Workshop on Business Process Intelligence (BPI 2011), volume 99 of Lecture Notes in Business Information Processing, pages 75-86. Springer-Verlag, Berlin (2012)