Algorithms & Theories for the Analysis of Event Data (ATAED'15, Brussels, Belgium, June 22-23, 2015)

Workshop Proceedings

Workshop on

Algorithms & Theories for the Analysis of Event Data (ATAED'2015)

Brussels, Belgium, June 22-23, 2015

Satellite event of the conferences

15th International Conference on Application of Concurrency to System Design (ACSD 2015)

36th International Conference on Application and Theory of Petri Nets and Concurrency (PN 2015)

Edited by
Wil van der Aalst, Robin Bergenthum, and Josep Carmona

These proceedings are published online by the editors as Volume 1371 at CEUR Workshop Proceedings

http://ceur-ws.org/Vol-1371

Preface

Regions have been defined about 20 years ago by Ehrenfeucht and Rozenberg as sets of nodes of a finite transition system that correspond to potential conditions that enable or disable transition occurrences in a corresponding elementary net system. Later, similar concepts were used to derive Petri nets from languages. Both *state-based* and *language-based* approaches aim to constrain a Petri net by adding places that are called *regions*. Over time, many variations have been proposed, e.g., approaches dealing with multiple-token in a place, extensions to partial orders, etc.

Initially, region theory focused on *synthesis* where the input behavior and resulting Petri net are supposed to be equivalent with respect to some equivalence criterion (e.g., bisimilar). Recently, region-based research started to focus also on *process mining* where the goal is *not* to create an equivalent model, but to *infer* new knowledge from the input. Process mining takes as input observed behavior rather than assuming a complete description in terms of a transition system or prefix-closed language. Classical region based techniques are unable to discover process models from event logs. One needs to deal with new problems such as noise and incompleteness. Equivalence notions are replaced by trade-offs between fitness, simplicity, precision, and generalization. A model with good *fitness* allows for most of the behavior seen in the event log. A model that does not *generalize* is "overfitting". Overfitting is the problem that a very specific model is generated whereas it is obvious that the log only holds example behavior. A model that allows for "too much behavior" lacks *precision* (i.e., is underfitting). Simplicity is related to Occam's Razor which states that "one should not increase, beyond what is necessary, the number of entities required to explain anything". Following this principle, we often look for the *simplest* process model that can explain what was observed in the event log. Process discovery from event logs is very challenging because of these and many other trade-offs. Clearly, there are many theoretical process-mining challenges with a high practical relevance that need to be addressed urgently.

The challenges and opportunities formed the main motivation for proposing the *Algorithms & Theories for the Analysis of Event Data* (ATAED'2015) workshop as a succession of the *Applications of Region Theory* (ART) workshop series. Our goal was (and is) to bring together researchers working on region-based synthesis and process mining. Looking at the proceedings, we succeeded in doing so!

The ATAED'2015 workshop took place in Brussels on June 22-23, 2015 and was a satellite event of both the 36th International Conference on Application and Theory of Petri Nets and Concurrency (Petri nets 2015) and the 15th International Conference on Application of Concurrency to System Design (ACSD 2015). Papers related to process mining, region theory and other synthesis techniques were presented at ATAED'2015. These techniques have in common that 'lower level' behavioral descriptions (event logs, partial orders, transition systems, etc.) are used to create 'higher level' process models (e.g., various classes of Petri nets, BPMN, or UML activity diagrams). In fact, all techniques that aim at learning or checking concurrent behavior from transition systems, runs, or event logs were welcomed. The workshop was supported by the IEEE Task Force on Process Mining (www.win.tue.nl/ieeetfpm/).

After a careful reviewing process, eleven papers were accepted for the workshop. Overall, the quality of the submitted papers was good and most submissions matched very well the workshop goals. We thank the reviewers for providing the authors with valuable and constructive feedback. Moreover, we were honored that *Eike Best* (University of Oldenburg) was willing to give an invited talk on the *"Synthesis of Diamonds"*. We thank Eike, the authors, and the presenters for their wonderful contributions.

In the remainder, the accepted papers of the Algorithms & Theories for the Analysis of Event Data (ATAED'2015) workshop are briefly summarized.

- The paper *"On Binary Words Being Petri Net Solvable"* by Kamila Barylska, Eike Best, Evgeny Erofeev, Lukacs Mikulski, and Marcin Piatkowski studies the class of two-letter Petri net solvable words, i.e., Petri nets with two transitions and a reachability graph isomorphic to a trace-based transition system. Several intriguing results are presented for this class, e.g., the existence of side-place-free solutions given particular conditions.
- Andrey Mokhov and Josep Carmona use Conditional Partial Order Graphs (CPOGs) to create compact and easy-to-comprehend visualizations of event logs with data. In their paper *"Event Log Visualization with Conditional Partial Order Graphs: From Control Flow to Data"*, the authors provide a technique to automatically derive the control-flow part of the CPOG representation from an event log, and then incorporate the data contained in the log as conditions for the CPOG vertices and arcs.
- The paper *"Discovery of Personal Processes from Labeled Sensor Data: An Application of Process Mining to Personalized Health Care"* by Timo Sztyler, Johanna Völker, Josep Carmona, Oliver Meier, and Heiner Stuckenschmidt shows how process mining can be used for analyzing self-tracking data. Smart-phones and smart-watches can be used to produce detailed data about someone's daily life. The authors describe the acquisition of such data in real-life and use existing process mining techniques for eliciting, analyzing and monitoring daily routines.
- *"ILP-Based Process Discovery Using Hybrid Regions"* by Sebastiaan van Zelst, Boudewijn Van Dongen, and Wil van der Aalst unifies the two existing types of language-based regions (single variable-based regions and dual variable-based regions) to provide a representation suitable for process mining. Integer Linear Programming (ILP)-based process discovery is further enhanced with a generalized ILP objective function. It is shown that any instantiation of the objective function leads to ILPs that favor minimal regions.
- Robin Bergenthum, Thomas Irgang, and Benjamin Meis present a folding algorithm to construct a business process model from a specification in their paper *"Folding Example Runs to a Workflow Net"*. Different to mainstream process mining techniques the input is not a sequential event log but a set of

example runs represented as labeled partial orders. By adopting ideas from the theory of regions, the authors aim at improving precision of the model while folding the runs into a model.

- The paper *"Mining Duplicate Tasks from Discovered Processes"* by Borja Vázquez-Barreiros, Manuel Mucientes, and Manuel Lama tackles the classical problem of label splitting in process mining. The authors propose an approach that uses the local information in the log to enhance an already mined model by performing a local search over the potential tasks to be duplicated. Experimental results show that, in a case study, the final model was improved in 35 out of 36 cases.
- The paper *"A Method For Assessing Parameter Impact on Control-Flow Discovery Algorithms"* by Joel Ribeiro and Josep Carmona presents a method to automatically assess the impact of parameters of control-flow discovery algorithms. The metaheuristic approach for process mining can be used to guide the user in selecting a technique, representational bias, and suitable parameter setting. The method has been evaluated over a set of logs while using the flexible heuristic miner.
- Antonia Azzini, Paolo Ceravolo, Ernesto Damiani, and Francesco Zavatartelli introduce the notion of extended behavior in their paper *"Knowledge Driven Behavioural Analysis in Process Intelligence"*. They present a methodology where first the descriptive knowledge is collected and the knowledge base queried, then (in the prescriptive and predictive knowledge phases) business rules and objectives are evaluated and unexpected business patterns and exceptions are uncovered.
- *"Compact Regions for Place/Transition Nets"* by Robin Bergenthum presents an approach using compact regions to synthesize a Petri net from a partial language. The language of a marked Petri net is its set of compact valid example runs. Compact regions are relevant as they may lead to faster synthesis algorithms computing smaller Petri nets. Initial results suggest that synthesis is indeed faster and that the compact solution space leads to nets having less places.
- In *"An Optimal Process Model for a Real Time Process"*, the authors Likewin Thomas, Manoj Kumar M.V., Annappa B., and Vishwanath K.P. provide a solution for recommending an optimal path of execution taking into account resource allocations. The proposed AlfyMiner compares variants of the same process encountered in different organizations. The authors include functionality to compare processes and to analyze resource behavior. This is then used to recommend next actions and suitable resources.
- The paper *"Capturing the Sudden Concept Drift in Process Mining"* by Manoj Kumar M.V., Likewin Thomas and Annappa B. focuses on sudden changes during process execution, i.e., second-order process dynamics. The paper proposes the extraction of a so-called "event class correlation feature" from logs for localizing the sudden concept drift in the control-flow perspective of the operational process. Experiments using synthetic event data show that (under ideal circumstances) sudden process changes can be detected.

The workshop proceedings provide a nice selection of ongoing research on the intersection of process mining and region-based synthesis. The papers illustrate the range of problems and solution approaches related to lifting 'lower level' dynamic behavior to 'higher level' process models. Given the rapid growth of event data, the area is expected to become even more relevant in years to come. We hope that ATAED'2015 serves as a starting point for a viable workshop series bringing together the two communities working on process mining and region-based synthesis.

Enjoy reading the proceedings!

Wil van der Aalst, Robin Bergenthum, and Josep Carmona
June 2015

Program committee of ATAED'2015

Wil van der Aalst, TU Eindhoven, The Netherlands (co-chair)
Rafael Accorsi, Universitaet Freiburg, Germany
Eric Badouel, INRIA Rennes, France
Robin Bergenthum, FernUni Hagen, Germany (co-chair)
Luca Bernardinello, Universit degli studi di Milano-Bicocca, Italy
Seppe vanden Broucke, KU Leuven, Belgium
Benoît Caillaud, INRIA Rennes, France
Toon Calders, Universit Libre de Bruxelles, Belgium
Josep Carmona, UPC Barcelona, Spain (co-chair)
Paolo Ceravolo, University of Milan, Italy
Benoît Depaire, Hasselt University, Belgium
Jörg Desel, FernUni Hagen, Germany
Boudewijn van Dongen, TU Eindhoven, The Netherlands
Luciano Garca-Bañuelos, University of Tartu, Estonia
Luís Gomes, Universidade Nova de Lisboa, Portugal
Gabriel Juhás, Slovak University of Technology, Slovak Republic
Anna Kalenkova, Higher School of Economics NRU, Russia
Jetty Kleijn, Leiden University, The Netherlands
Robert Lorenz, Uni Augsburg, Germany
Zbigniew Paszkiewicz, PricewaterhouseCoopers, Belgium
Marta Pietkiewicz-Koutny, Newcastle University, GB
Grzegorz Rozenberg, Leiden University, The Netherlands
Marcos Sepúlveda, Universidad Catolica de Chile, Chile
Jianmin Wang, Tsinghua University, China
Jochen De Weerdt, KU Leuven, Belgium
Alex Yakovlev, Newcastle University, GB

Table of Contents

On Binary Words Being Petri Net Solvable

Kamila Barylska[1,2⋆†], Eike Best[1∗,⋆⋆], Evgeny Erofeev[1⋆⋆⋆], Łukasz Mikulski[2†],
Marcin Piątkowski[2†]

[1] Department of Comp. Sci., Carl von Ossietzky Univ. Oldenburg, Germany
{eike.best,evgeny.erofeev}@informatik.uni-oldenburg.de
[2] Faculty of Math. and Comp. Sci., Nicolaus Copernicus University Toruń, Poland
{kamila.barylska,lukasz.mikulski,marcin.piatkowski}@mat.umk.pl

Abstract. A finite word is called Petri net solvable if it is isomorphic to the reachability graph of some unlabelled Petri net. In this paper, the class of two-letter Petri net solvable words is studied.

Keywords: Binary words, labelled transition systems, Petri nets, synthesis

1 Introduction

Region theory [1] provides a polynomial algorithm, based on solving linear inequations, that checks whether a given finite labelled transition system is the reachability graph of some place/transition Petri net [5], and if it is, synthesises one of them. Due to the size of a transition system, this algorithm may be very time-consuming. Moreover, it may produce one out of a class of different nets, and there may not be a unique simplest one. For some applications, only certain types of labelled transition systems are relevant. This leads to the idea of investigating properties of labelled transition systems before synthesising them, in the hope of obtaining more efficient and possibly also more deterministic synthesis algorithms. It may even be possible to find exact structural characterisations, based solely on graph-theoretical properties, such as for the class of finite labelled transition systems which correspond to T-systems [2].

This paper reports progress on a similar effort about characterising the set of finite words over an alphabet $\{a, b\}$ which are Petri net solvable, i.e., for which a place/transition net with an isomorphic reachability graph exists. We shall put forward two conjectures, and describe some progress in analysing them. This

[⋆] Supported by a research and a visiting fellowship within the project "Enhancing Educational Potential of Nicolaus Copernicus University in the Disciplines of Mathematical and Natural Sciences" (project no. POKL.04.01.01-00-081/10)

[⋆⋆] Supported by the German Research Foundation through the DFG/RFBR cooperation project CAVER, BE 1267/14-1

[⋆⋆⋆] Supported by the German Research Foundation through the Research Training Group (DFG GRK 1765) SCARE – http://www.uni-oldenburg.de/en/scare/

[†] Supported by the National Science Center under grant no. 2013/09/D/ST6/03928

work could well be of interest in a wider context, as it might entail a nontrivial necessary condition for the solvability of an arbitrary labelled transition system. If the latter is solvable, then finding a PN-unsolvable word as a path in it may have a strong impact on its structure / shape.

2 Basic notations and conventions used in this paper

A *finite labelled transition system* with initial state is a tuple $TS = (S, \to, T, s_0)$ with nodes S (a finite set of states), edge labels T, edges $\to \subseteq (S \times T \times S)$, and an initial state $s_0 \in S$. A label t is enabled at $s \in S$, written formally as $s[t\rangle$, if $\exists s' \in S \colon (s, t, s') \in \to$. A state s' is reachable from s through the execution of $\sigma \in T^*$, denoted by $s[\sigma\rangle s'$, if there is a directed path from s to s' whose edges are labelled consecutively by σ. The set of states reachable from s is denoted by $[s\rangle$. A (firing) sequence $\sigma \in T^*$ is allowed from a state s, denoted by $s[\sigma\rangle$, if there is some state s' such that $s[\sigma\rangle s'$. Two lts $TS_1 = (S_1, \to_1, T, s_{01})$ and $TS_2 = (S_2, \to_2, T, s_{02})$ are isomorphic if there is a bijection $\zeta \colon S_1 \to S_2$ with $\zeta(s_{01}) = s_{02}$ and $(s, t, s') \in \to_1 \Leftrightarrow (\zeta(s), t, \zeta(s')) \in \to_2$, for all $s, s' \in S_1$.

A *word over* T is a sequence $w \in T^*$, and it is *binary* if $|T| = 2$. A word $t_1 t_2 \ldots t_n$ of length $n \in \mathbb{N}$ uniquely corresponds to a finite transition system $(\{0, \ldots, n\}, \{(i-1, t_i, i) \mid 0 < i \leq n \wedge t_i \in T\}, T, 0)$.

An *initially marked Petri net* is denoted as $N = (P, T, F, M_0)$ where P is a finite set of places, T is a finite set of transitions, F is the flow function $F \colon ((P \times T) \cup (T \times P)) \to \mathbb{N}$ specifying the arc weights, and M_0 is the initial marking (where a marking is a mapping $M \colon P \to \mathbb{N}$, indicating the number of tokens in each place). A side-place is a place p with $p^\bullet \cap {}^\bullet p \neq \emptyset$, where $p^\bullet = \{t \in T \mid F(p,t) > 0\}$ and ${}^\bullet p = \{t \in T \mid F(t,p) > 0\}$. N is pure or side-place free if it has no side-places. A transition $t \in T$ is enabled at a marking M, denoted by $M[t\rangle$, if $\forall p \in P \colon M(p) \geq F(p,t)$. The firing of t leads from M to M', denoted by $M[t\rangle M'$, if $M[t\rangle$ and $M'(p) = M(p) - F(p,t) + F(t,p)$. This can be extended, as usual, to $M[\sigma\rangle M'$ for sequences $\sigma \in T^*$, and $[M\rangle$ denotes the set of markings reachable from M. The reachability graph $RG(N)$ of a bounded (such that the number of tokens in each place does not exceed a certain finite number) Petri net N is the labelled transition system with the set of vertices $[M_0\rangle$, initial state M_0, label set T, and set of edges $\{(M, t, M') \mid M, M' \in [M_0\rangle \wedge M[t\rangle M'\}$. If an lts TS is isomorphic to the reachability graph of a Petri net N, we say that N *solves* TS.

3 Separation problems, and an example

In region theory, a labelled transition system $(S, \to, T, s_0)$ is assumed to be given as an input. In order to synthesise (if possible) a Petri net with isomorphic reachability graph, T is used as the set of transitions, and for the places, $\frac{1}{2} \cdot (|S| \cdot (|S| - 1))$ *state separation problems* and up to $|S| \cdot |T|$ *event/state separation problems* have to be solved. A state separation problem consists of a set of states $\{s, s'\}$ with $s \neq s'$, and for each such set, one needs a place that distinguishes them. Such problems are always solvable for words; for instance, we might introduce a counting place which simply has j tokens in state j. An event/state separation problem consists of a pair $(s, t) \in S{\times}T$ with $\neg(s[t\rangle)$. For every such problem, we need a place p such that $M(p) < F(p, t)$ for the marking M corresponding to state s, where F refers to the arcs of the hoped-for net.

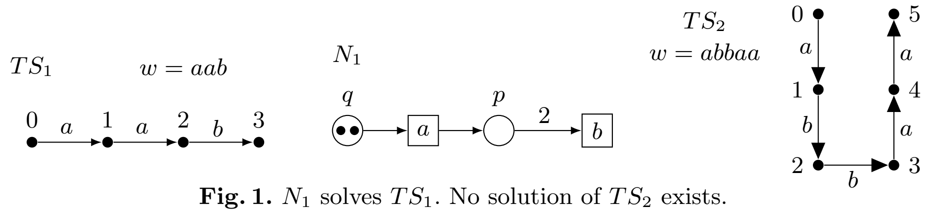


Fig. 1. N_1 solves TS_1. No solution of TS_2 exists.

For example, in figure 1, the labelled transition systems TS_1 and TS_2 correspond to the words *aab* and *abbaa*, respectively. The former is PN-solvable, since the reachability graph of N_1 is isomorphic to TS_1. TS_2 contains an unsolvable event/state separation problem. The state $s = 2$ just between the two b's satisfies $\neg(s[a\rangle)$. We need a place q whose number of tokens in (the marking corresponding to) state 2 is less than needed for transition a to be enabled. Such a place q has the general form shown on the right-hand side of figure 2. It is useful to speak of the *effect* $\mathbb{E}(\tau)$ of a sequence $\tau \in T^*$ on place q. For the letter a, the effect is defined as $\mathbb{E}(a) = (a_+ - a_-)$, and this can be generalised easily. Thus, for instance, the effect $\mathbb{E}(abbaa)$ is $\mathbb{E}(abbaa) = 3{\cdot}(a_+ - a_-) + 2{\cdot}(b_+ - b_-)$. If q prevents a at state 2 in *abbaa*, then it must satisfy the following inequalities, amongst others: $a_- \leq m$, since a is enabled initially; $a_- \leq m + \mathbb{E}(abba)$, since state 4 enables a; and $m + \mathbb{E}(ab) < a_-$, or equivalently, $0 \leq -m - \mathbb{E}(ab) + a_- - 1$, expressing the fact that q solves the event/state separation problem $\neg(2[a\rangle)$. Later, we show that this set of inequalities cannot be solved in the natural numbers.

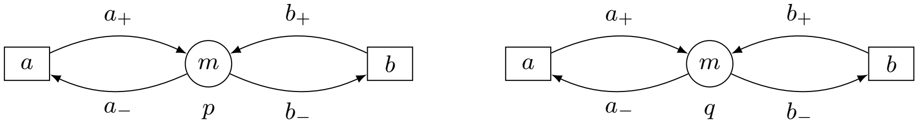


Fig. 2. Places p / q with four arc weights a_-, a_+, b_-, b_+ and initial marking m. They are similar, but p will be used for preventing b, and q for preventing a.

In a word of length n, the equation system for a single event/state separation problem comprises $n + 1$ inequations. In binary words, we have $n + 2$ such problems, one for every state $0, \ldots, n - 1$ and two for the last state. Thus, a word w of length n is PN-solvable if and only if all those $n + 2$ systems, each having $n + 1$ inequalities and five unknowns a_-, a_+, b_-, b_+, m, are solvable in $\mathbb{N}$. The question dealt with in this paper is whether the set of binary words that are PN-(un)solvable can be characterised equivalently, in a more structural way. We shall assume, from now on, that $T = \{a, b\}$.

4 Minimal unsolvable binary words, and some conjectures

Let a word $w' \in T^*$ be called a *subword* (or *factor*) of $w \in T^*$ if $\exists(u_1, u_2 \in T^*)$: $w = u_1 w' u_2$, and let $\#_t(w)$ denote the number of times the letter t occurs in w. Observe that if w is PN-solvable then all its subwords are, too. To see this, let the Petri net solving w be executed up to the state before w', take this as the new initial marking, and add a pre-place with $\#_a(w')$ tokens to a and a pre-place with $\#_b(w')$ tokens to b. Thus, if a subword of w is unsolvable, then w is. For this reason, the notion of a *minimal unsolvable word* is well-defined (namely, as an unsolvable word all of whose subwords are solvable). A complete list of minimal unsolvable words up to length 110 can be found in [6]. As a consequence of the next proposition, any minimal unsolvable word either starts and ends with a or starts and ends with b.

Proposition 1. Solvability of aw and wb implies solvability of awb

If both aw and wb are solvable, then awb is also solvable.

Proof: Assume that aw and wb are PN-solvable words over $\{a, b\}$. If $w = b^k$ (for $k \in \mathbb{N}$) then $awb = ab^{k+1}$ is obviuosly solvable, hence we assume that b contains at least one a. Let $N_1 = (P_1, \{a, b\}, F_1, M_{01})$ and $N_2 = (P_2, \{a, b\}, F_2, M_{02})$ be Petri nets such that N_1 solves aw and N_2 solves wb. We can assume that N_1 and N_2 are disjoint, except for their transitions a and b. Forming the union of N_1 and N_2 gives a net which is synchronised at a and b, and which allows all (and only) sequences allowed by both N_1 and N_2. We modify N_1 and N_2 before forming their union, as follows:

(i) Modify N_1 by adding, to each place p in ${}^\bullet b \cap P_1$, another $F_1(p, b)$ tokens. This allows an additional b.

(ii) Modify N_2 by adding, to each place q in ${}^\bullet a \cap P_2$, another $F_2(q, a)$ tokens. This allows an additional a. Further, for each place p in $a^\bullet \cap P_2 \cap {}^\bullet b$, add the quantity $F_2(a, p)$ both to $F_2(p, b)$ and to $F_2(b, p)$. The new arc weights lead to the same effect of b on p, but prevent premature occurrences of b which could have been allowed by adding the tokens in front of b in step (i).

Define N as the union of the two nets thus modified, and see figure 3 for an example. (The added tokens are drawn as hollow circles.) In general, N solves awb in the following way: The initial a is allowed in N_1 by definition and in N_2 by the additional tokens. The subsequent w is allowed in both nets, and hence in their synchronisation. The final b is allowed in N_2 by definition and in N_1 by the additional tokens. No premature b is allowed by the arc weight increase, and no final additional a is allowed because N_1 does not allow it. □ 1

¿From the list [6], it can be observed that all minimal unsolvable words starting and ending with a are of the following general form:

$$s_0 \; [\; (ab\alpha)\, b^* \;\rangle\; s \; [\; (ba\alpha)^+ \;\rangle\; r \; [\; a \;\rangle \quad \text{where } \alpha \in T^* \text{ and } s_0,\, s,\, r \text{ are states} \quad (1)$$

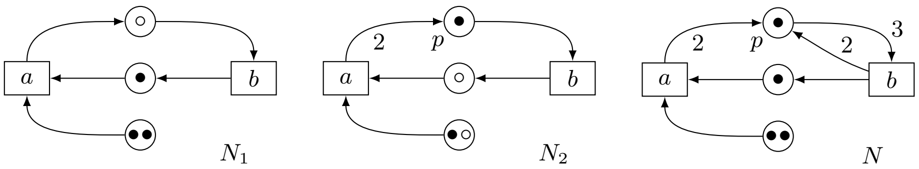


Fig. 3. N_1 (black tokens) solves $aw = abab$. N_2 (black tokens) solves $wb = babb$. N (redundant places omitted) solves $awb = ababb$.

with a not being separated at s. For example, $abbaa$ satisfies (1) with $\alpha = \varepsilon$, the star * being repeated zero times, and the plus $^+$ being repeated just once. Indeed, it is easy to prove that such words are generally PN-unsolvable:

Proposition 2. Sufficient condition for the unsolvability of a word
If a word over $\{a, b\}$ has a subword of the form (1)*, then it is not PN-solvable.*

Proof: For a word w of the form (1), we prove that w cannot be solved (implying the proposition in the context of the considerations above). Because $ba\alpha$ occurs at least once after state s, b is enabled at s, and a is not enabled at s. Suppose that some place q as in figure 2 (r.h.s.) exists which separates a at s. Let E be $\mathbb{E}(ab\alpha)$, i.e., the effect of $ab\alpha$ on q, and let $E_b = \mathbb{E}(b)$. $\mathbb{E}(b) \geq 1$ because q separates a at s but not at $s + 1$. For w, we derive the following inequalities:

$$\begin{array}{rlll}
(0) & a_- & \leq m & \\
(s{+}1) & a_- & \leq m + E + k{\cdot}E_b + E_b & \text{for some fixed } k \geq 0 \\
(r) & a_- & \leq m + E + k{\cdot}E_b + \ell{\cdot}E & \text{for the same } k \text{ and some fixed } \ell > 0 \\
(\text{sep}) & 0 & \leq -m - E - k{\cdot}E_b + a_- - 1 & \text{for the same } k
\end{array}$$

(0) is true because at s_0, a is enabled. (s+1) is true because a is enabled one state after s. (r) is true because a is enabled at r; and $\ell > 0$ because of the $^+$. Finally, (sep) is true because q disables a at state s. Adding (s+1)+(sep) gives $1 \leq E_b$. Adding (0)+(sep) gives $1 \leq -E - k{\cdot}E_b$, and using also $1 \leq E_b$ gives $1 \leq -E - k{\cdot}E_b \leq -E$. Adding ($r$)+(sep) gives $1 \leq \ell{\cdot}E$, contradicting $1 \leq -E$. The system cannot be solved, and no place q separating a at s exists. □ 2

Conjecture 1. A converse of proposition 2
Suppose a word over $\{a, b\}$ is non-PN-solvable and minimal with that property. Then it is (modulo swapping a and b) of the form given in (1). *Strengthened conjecture*: It is either of the form $ab\underbrace{b^j}_{\alpha}b^kba\underbrace{b^j}_{\alpha}a$ with $j \geq 0, k \geq 1$ or of the form $ab\alpha(ba\alpha)^\ell a$ with $\ell \geq 1$ □ Conj. 1

Proposition 3. Another sufficient condition for unsolvability
Let w be of the form $s_0[\alpha\rangle s[\beta\rangle r[a\rangle$ such that α starts with a and β starts with b.

$$\textit{If} \quad \#_a(\beta){\cdot}\#_b(\alpha) \;\geq\; \#_a(\alpha){\cdot}\#_b(\beta) \tag{2}$$

then w is unsolvable.

For instance, for $w = abbaa$, $\alpha = ab$ and $\beta = ba$, and (2) holds true.

Proof: If a place q separates a at s and has marking m at s_0, then for $E_\alpha = \mathbb{E}(\alpha) = \#_a(\alpha){\cdot}E_a + \#_b(\alpha){\cdot}E_b$ and $E_\beta = \mathbb{E}(\beta) = \#_a(\beta){\cdot}E_a + \#_b(\beta){\cdot}E_b$ we have:

$$\begin{array}{rlll}
(0) & a_- & \leq m & \text{(since } \alpha \text{ starts with } a\text{)} \\
(r) & a_- & \leq m + E_\alpha + E_\beta & \\
\text{(sep)} & 0 & \leq -m - E_\alpha + a_- - 1 & \text{(since } \neg s[a\rangle\text{)}
\end{array}$$

Adding (0)+(sep) yields $1 \leq -E_\alpha$, hence (A): $-(\#_a(\alpha)E_a + \#_b(\alpha)E_b) \geq 1$, where E_a and E_b denote the effects of a and b on q, respectively. As before, $E_b \geq 1$. Adding (r)+(sep) yields $1 \leq E_\beta$, hence (B): $(\#_a(\beta)E_a + \#_b(\beta)E_b) \geq 1$. Then,

$$\begin{array}{rcll}
-\#_a(\beta) & \geq & \#_a(\beta)\#_a(\alpha)E_a + \#_a(\beta)\#_b(\alpha)E_b & \text{(algebra, and by (A))} \\
 & \geq & \#_a(\beta)\#_a(\alpha)E_a + \#_a(\alpha)\#_b(\beta)E_b & \text{(using (2) and } E_b \geq 1\text{)} \\
 & \geq & \#_a(\alpha) & \text{(algebra, and by (B))}
\end{array}$$

However, $-\#_a(\beta) \geq \#_a(\alpha)$ implies $\#_a(\beta) = \#_a(\alpha) = 0$, and this is a contradiction since α contains at least one a. Thus, such a place q does not exist. □ 3

Words in which the letters a and b strictly alternate are easy to solve. Therefore, it stands to reason to investigate cases in which a letter occurs twice in a row.

Proposition 4. Solvable words starting with a can be prefixed by a

If a word av is PN-solvable then aav is, too.

Proof: Let $N = (P, \{a, b\}, F, M_0)$ be a net solving av. We shall construct a net which solves aav. The idea is to obtain such a net by "unfiring" a once from the initial marking of N. Since this may lead to a non-semipositive marking which we would like to avoid, we will first normalise and modify the net N, obtaining another solution N' of av, and then construct a solution N'' for aav (cf. Fig. 4).

For normalisation, we assume that there are two places p_b and q_a; the first prevents b explicitly in the initial phase, and the second prevents a after the last occurrence of a. They are defined by $M_0(p_b) = 1$, $F(a, p_b) = 1$, $F(b, p_b) = \ell{+}1 = F(p_b, b)$, where ℓ is the number of a before the first b in av, and $M_0(q_a) = k$, $F(q_a, a) = 1$, where k is the number of a in av. (All other F values $= 0$.)

Let $NUF(a) = \{p \in a^\bullet \mid M_0(p) < F(a, p)\}$ be the set of places which do not allow the "unfiring" of a at M_0. Note that neither p_b nor q_a are in $NUF(a)$. Note also that for every $p \in NUF(a)$, $F(p, a) \leq M_0(p) < F(a, p)$ – the first because a is initially enabled, the second by $p \in NUF(a)$. That is, a has a positive effect on p. Without loss of generality, b has a negative effect on p (otherwise, thanks to the normalising place p_b, p could be deleted without changing the behaviour of N).

For every $p \in NUF(a)$ we add the quantity $F(a, p)$ uniformly to $M_0(p)$, to $F(p, b)$, and to $F(b, p)$, eventually obtaining $N' = (P', \{a, b\}, F', M_0')$, and we show that N' also solves av. First, both $M_0[a\rangle \wedge \neg M_0[b\rangle$ and $M_0'[a\rangle \wedge \neg M_0'[b\rangle$ (the former by definition, the latter by construction). For an inductive proof, suppose that $M_0[a\rangle M_1[\tau\rangle M$ and $M_0'[a\rangle M_1'[\tau\rangle M'$. We have $M[b\rangle$ iff $M'[b\rangle$ by construction. If

$M[a\rangle$, then also $M'[a\rangle$, since $M \leq M'$. Next, suppose that $\neg M[a\rangle$; then there is some place q such that $M(q) < F(q,a)$. We show that, without loss of generality, $q \notin NUF(a)$, so that q also disables a at M' in N'. If M disables the last a in av, we can take $q = q_a \notin NUF(a)$. If M disables some a which is not the last one in av, then q cannot be in $NUF(a)$, since b acts negatively on such places.

Now, we construct a net $N'' = (P', \{a,b\}, F', M_0'')$ from N' by defining $M_0''(p) = M_0'(p) - F'(a,p) + F'(p,a)$ for every place p. By construction, aav is a firing sequence of N''. Furthermore, M_0'' does not enable b because of p_b. □ 4

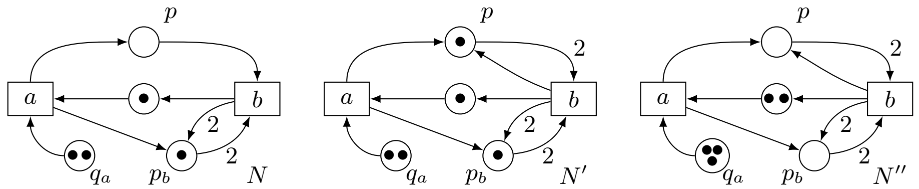


Fig. 4. N is normalised and solves $abab$. N' solves $abab$ as well. N'' solves $aabab$.

As a consequence, if av is minimally PN-unsolvable, then v starts with a b.

Proposition 5. No aa and bb inside a minimal unsolvable word

If a minimal non-PN-solvable word is of the form $u = a\alpha a$, then either α does not contain the factor aa or α does not contain the factor bb.

Proof: By contraposition. Assume that α contains a factor aa and a factor bb. Two cases are possible:

Case 1: There is a group of a's which goes after a group of b's. Let a^m and b^n be such groups, assume that a^m goes after b^n and that there are no groups of a or of b between them. Then u is of the following form

$$s_0\ [\ \dots\ \rangle\ q\ [\ ab^n(ab)^k a^m\ \rangle\ r\ [\ \dots\ \rangle$$

where $n, m \geq 2,\ k \geq 0$. Recombine the letters in u to the following form:

$$s_0\ [\ \dots\ \rangle\ q\ [\ (ab)b^{n-2}(ba)^{k+1}aa^{m-2}\ \rangle\ r\ [\ \dots\ \rangle$$

Since u ends with a, $(ab)b^{n-2}(ba)^{k+1}a$ is a proper subword of u. But it has the form $(abw)b^*(baw)^+a$, with $w = \varepsilon$, which implies its unsolvability by proposition 2, contradicting the minimality of u.

Case 2: All groups of a precede all groups of b. In this case u is of the form

$$aa^{x_0}ba^{x_1}\dots ba^{x_n}b^{y_0}ab^{y_1}ab^{y_2}\dots ab^{y_m}a$$

where at least one of x_i and one of y_j is greater than 1. Consider $\ell = \max\{i \mid x_i > 1\}$. If $\ell = 0$, we get a contradiction to proposition 4. Hence, $\ell > 0$. Let $t = \min\{j \mid y_j > 1\}$. Then u has the form

$$s_0\ [\ a\dots\ \rangle\ q\ [\ ba^{x_\ell}(ba)^{n-\ell}(ba)^t b^{y_t}\ \rangle\ r\ [\ \dots a\ \rangle$$

Recombine the letters in u to the form

$$s_0\ [\ a\dots\ \rangle\ q\ [\ (ba)a^{x_\ell-2}(ab)^{n-\ell+t+1}bb^{y_t-2}\ \rangle\ r\ [\ \dots a\ \rangle$$

Hence, u has a proper subword $(ba)a^{x_\ell-2}(ab)^{n-\ell+t+1}b$, which is of the form $(baw)a^*(abw)^+b$ with $w=\varepsilon$, implying its non-PN-solvability, due to proposition 2 with inverted a and b. This again contradicts the minimality of u. □ 5

For these reasons, we are particularly interested in words of the following form:

$$\text{either}\quad ab^{x_1}a\dots ab^{x_n}a \quad\text{or}\quad b^{x_1}a\dots ab^{x_n} \qquad \text{where } x_i\geq 1 \text{ and } n>1 \tag{3}$$

In the first form, there are no factors aa. If factors bb are excluded and the word starts and ends with an a, then we get words that are of the second form, except for swapping a and b. This swapping is useful in order to understand how words of the two forms are interrelated.

Conjecture 2. A CONVERSE OF PROPOSITION 3
If a word is of the form $w=\alpha\beta a$ where α starts with a and β starts with b, and if w is minimal non-PN-solvable and also of the form given in (3), then inequation (2) holds. *A stronger variant of this conjecture*: If $w=\alpha\beta a$ is of the form

$$w\quad=\quad [\ \underbrace{ab^{x_1}a\dots ab^{x_k-1}}_{\alpha}\ \rangle\ s\ [\ \underbrace{ba\dots ab^{x_n}}_{\beta}\ a\ \rangle \qquad \text{with } n\geq 3 \text{ and } x_i\geq 1$$

then a is not separated at state s iff $\#_a(\beta)\cdot\#_b(\alpha)=\#_a(\alpha)\cdot\#_b(\beta)$. □ Conj. 2

5 Some results about words of the form $b^{x_1}a\dots ab^{x_n}$

5.1 Side-places in words of the form $b^{x_1}a\dots ab^{x_n}$

If a word $w=b^{x_1}a\dots ab^{x_n}$ can be solved at all, then side-places may be necessary to do it. However, we will show that in the worst case, only some side-place q around a, preventing a at some state, are necessary.

Lemma 1. SIDE-PLACE-FREENESS AROUND b
If $w=b^{x_1}a\dots ab^{x_n}$ is solvable, then w is solvable without side-place around b.

Proof: Let w and its intermediate states be of the form[1]

$$w\quad=\quad s_0[b^{x_1}\rangle s_1[ab^{x_2}\rangle s_2[a\rangle\dots s_{n-1}[ab^{x_n}\rangle s_n \tag{4}$$

Suppose some place p prevents b at some state s_k, for $1\leq k\leq n-1$. (The only other state at which b must be prevented is state s_n, but that can clearly be done

[1] A note on convention: in the following, we use the letter s to denote states at which b has to be prevented, and p for places doing this. Similarly, we use the letter r to denote states at which a has to be prevented, and q for places doing this.

by a non-side-place, e.g. by an incoming place of transition b that has initially $\sum_{i=1}^{n} x_i$ tokens.) Note that $b_- > b_+$, because place p allows b to be enabled at the state preceding s_k but not at s_k. Similarly, $a_- < a_+$, because b is not enabled at state s_k but at the immediately following state, which is reached after firing a. ¿From the form (4) of w, we have

$$\begin{array}{ll} b_+ & \leq m + x_1(b_+ - b_-) \\ b_+ & \leq m + (x_1 + x_2)(b_+ - b_-) + (a_+ - a_-) \\ \cdots & \\ b_+ & \leq m + (x_1 + \ldots + x_n)(b_+ - b_-) + (n-1)(a_+ - a_-) \\ 0 & \leq -m - (x_1 + \ldots + x_k)(b_+ - b_-) - (k-1)(a_+ - a_-) + b_- - 1 \end{array} \tag{5}$$

The first n inequations assert the semipositivity of the marking of place p (more precisely, its boundedness from below by b_+, since p may be a side-place) at the n states $s_1, \ldots, s_n$. In our context, if these inequalities are fullfilled, then the marking is $\geq b_+$ at *all* states, as a consequence of $b_- \geq b_+$, $a_- \leq a_+$, and the special form of the word. The last inequality comes from $\neg(s_k[b\rangle)$.

We certainly have $0 \leq b_+ \leq b_- \leq m$, because of $b_- \geq b_+$ as noted above, and because b is initially enabled. If $b_+ = 0$, then p is not a side-place around b, and there is nothing more to prove (for p). If $b_+ \geq 1$, we consider the transformation

$$b'_+ = b_+ - 1 \text{ and } b'_- = b_- - 1 \text{ and } m' = m - 1$$

The relation $0 \leq b'_+ \leq b'_- \leq m'$ still holds for the new values. Also, all inequalities in (5) remain true for the new values: in the first n lines, 1 is subtracted on each side, and on the last line, the increase in $-m$ is offset by the decrease in b_-. Thus, we get a solution preventing b with a 'smaller' side-place, and we can continue until eventually b_+ becomes zero.

A side-place around b might, however, still be necessary to prevent a at some state. We show next that such side-places are also unnecessary.

Let w and its intermediate states be of the form

$$[b^{x_1-1}\rangle r_1[b\,a\,b^{x_2-1}\rangle r_2[b\,a\,\rangle \ldots [\,a\,b^{x_k-1}\rangle r_k[b\rangle s_k[\,a\,\rangle \ldots [b^{x_{n-1}-1}\rangle r_{n-1}[b\,a\,b^{x_n}\rangle$$

Suppose some place q as on the right-hand side of figure 2 prevents a at state r_k, for $1 \leq k \leq n-1$. Symmetrically to the previous case, we have $b_+ > b_-$. This is true because, while q does not have enough tokens to enable a at state r_k, it must have enough tokens to enable a at the directly following state (which we may continue to call s_k). But we also have (w.l.o.g.) $a_+ < a_-$. For $k \geq 2$, this follows from the fact that if the previous a (enabled at the state s_{k-1} just after r_{k-1}) acts positively on q, then q also has sufficiently many tokens to enable a at state r_k. For $k = 1$, it is possible to argue that $a_+ \leq a_-$ is valid without loss of generality. For suppose that q disables a only at r_1 and nowhere else. (This is no loss of generality because for the other states r_k, $k \geq 2$, copies of q can be used.) Then we may consider q' which is an exact copy of q, except that $a_+ = a_- - 1$ for q'. This place q' also disables a at state r_1 (because it has the same marking as

q). Moreover, it does not disable a at any other state after r_1 because it always has $\geq a_- - 1$ tokens, and after the next b, $\geq a_-$ tokens, since $b_+ > b_-$.

Because of $b_+ \geq b_-$ and $a_+ \leq a_-$, q also prevents a at all prior states in the same group of b's. Moreover, in the last (i.e. n'th) group of b's, a can easily be prevented side-place-freely. For place q with initial marking m, we have

$$\begin{array}{ll} a_+ & \leq m + x_1(b_+ - b_-) + (a_+ - a_-) \\ a_+ & \leq m + (x_1 + x_2)(b_+ - b_-) + 2(a_+ - a_-) \\ \ldots & \\ a_+ & \leq m + (x_1 + \ldots + x_{n-1})(b_+ - b_-) + (n-1)(a_+ - a_-) \\ 0 & \leq -m - (x_1 + \ldots + x_k - 1)(b_+ - b_-) - (k-1)(a_+ - a_-) + a_- - 1 \end{array} \qquad (6)$$

The first $n-1$ inequations assert the semipositivity of the marking of place q (more precisely, its boundedness from below by a_+, since q may be a side-place of a) at the $n-1$ states $r_1, \ldots, r_{n-1}$. If these inequalities are fullfilled, then the marking is $\geq a_+$ at *all* states after the first a, as a consequence of $b_+ \geq b_-$ and the special form of the word. The last inequality asserts that place q prevents transition a at state r_k, hence effects the event/state separation of a at r_k.

If b_- is already zero, place q is not a side-place of b. Otherwise, we may perform the transformation

$$b'_+ = b_+ - 1 \text{ and } b'_- = b_- - 1 \text{ and } m' = m$$

because of $b_+ \geq b_-$ as noted above. The left-hand sides of the first $n-1$ inequalities in (6) do not decrease, and neither do the right-hand sides. The same is true for the last inequality. □ 1

Lemma 2. Side-place-freeness around a, preventing b

Suppose $w = b^{x_1}ab^{x_2}a \ldots ab^{x_n}$. If w is solvable by a net in which some place p separates b, then we may w.l.o.g. assume that p is not a side-place around a.

Proof: The equation system (5) is invariant under the transformation

$$a'_+ = a_+ - 1 \text{ and } a'_- = a_- - 1 \text{ and } m' = m$$

as neither the left-hand sides nor the right-hand sides change their values. □ 2

If some place q prevents transition a, then a side-place q connected to a may be present. It may not always be possible to remove such a side-place. Consider, for instance, the word $w = bbbabab$. It is of the form (4), and any net solving $bbbabab$ necessarily contains a side-place around transition a.[2] The word $bbabbababab$ can also not be solved without a side-place (but $bbabbabab$ can). So far, no tight (weak) sufficient conditions for solvability, or solvability without side-places around a, are known. However, the next lemma shows that the presence of a side-place around a may be due to there being "many" initial b's. That is, if x_1 is "small enough", then such a side-place is not necessary.

[2] The reader is invited to use [7] in order to verify this claim; we will not include a proof in this paper for lack of space.

Lemma 3. Side-place-freeness around a, preventing a

Suppose $w = b^{x_1}ab^{x_2}a \dots ab^{x_n}$. If $x_1 \leq \min\{x_2, \dots, x_{n-1}\}$ and if w is solvable by a net in which some place q prevents transition a at state r_k with $1 \leq k \leq n$, then we may w.l.o.g. assume that q is not a side-place around a.

Proof: For preventing a at state r_n, we only need a place with no input and output a (weight 1) which has $n-1$ tokens initially.

Suppose q prevents a at state r_k, with $1 \leq k \leq n-1$. ¿From previous considerations, we know $a_+ < a_-$ and $b_+ > b_-$, and we may assume, w.l.o.g., that q is not a side-place around b, i.e., that $b_- = 0$. The initial marking m of q and the remaining arc weights a_+, a_-, b_+ satisfy the system of inequations (6), except that it is simplified by $b_- = 0$. If $a_+ = 0$, then q is already of the required form. For $a_+ > 0$, we distinguish two cases.

Case 1: $m > 0$ and $a_+ > 0$. Then consider the transformation

$$m' = m - 1 \text{ and } a'_+ = a_+ - 1 \text{ and } a'_- = a_- - 1$$

By $m > 0$ and $a_- \geq a_+ > 0$, we get new values $m', a'_+, a'_- \geq 0$. Moreover, (6) remains invariant under this transformation. So, q' serves the same purpose as q, and it has one incoming arc from a less than q. By repeating this procedure, we either get a place which serves the same purpose as q, or we hit Case 2.

Case 2: $m = 0$ and $a_+ > 0$. In this case, we consider the transformation

$$m' = m = 0 \text{ and } a'_+ = 0 \text{ and } a'_- = a_-$$

Such a transformation also guarantees $m', a'_+, a'_- \geq 0$. Also, the last line of (6) is clearly satisfied with these new values, since the value of its right-hand stays the same (for $k = 1$) or increases (for $k > 1$). To see that the first $n-1$ lines of (6) are also true with the new values (and with $b_- = 0$), and that we can, therefore, replace q by q', we may argue as follows. At any marking $\widetilde{m}$ reached along the execution of w, we have the following:

$$\widetilde{m}(q) \quad \geq \quad \widetilde{m}(q') \quad \geq \quad 0 \tag{7}$$

These inequalities imply that the new place q' prevents a at r_k, whenever the old one, q, does, and that, moreover, no occurrences of a are excluded by the place q' where they should not be prohibited.

The first of the inequalities (7) holds because it holds initially (when $\widetilde{m} = m$, then $\widetilde{m}(q) = m = m' = \widetilde{m}(q')$), and because the effect of a before the transformation is $(a_+ - a_-)$, and after the transformation, it is $(-a_-)$. In other words, a reduces the token count on q' more than it does so on q, while b has the same effect on q' as on q. To see the second inequality in (7), let $x = \min\{x_2, \dots, x_{n-1}\}$. Then

$$a_- \quad \leq \quad x_1 \cdot b_+ \quad \leq \quad x \cdot b_+$$

The first inequality follows because $m = 0$ and q has enough tokens after the first x_1 occurrences of b in order to enable a. The second inequality follows from

$x_1 \leq x$. But then, since a only removes a_- tokens from q' and the subsequent block of b's puts at least $x \cdot b_+$ tokens back on q', the marking on q' is always ≥ 0, up to and including the last block of b's. □ 3

Corollary 1. Side-place-free solvability with few initial b's

If $w = b^{x_1}ab^{x_2}a \ldots ab^{x_n}$ is solvable, then side-places are necessary, at worst, between a and q, where q is some place preventing a at one of the states r_k with $1 \leq k < n-1$. If $w = b^{x_1}ab^{x_2}a \ldots ab^{x_n}$ is solvable and $x_1 \leq \min\{x_2, \ldots, x_{n-1}\}$, then w is solvable side-place-freely. □ 1

5.2 Solving words aw from words of the form $w = b^{x_1}a \ldots ab^{x_n}$

Solving a word of the form $w = b^{x_1}a \ldots ab^{x_n}$ side-place-freely allows us to draw some conclusion about prepending a letter a to it, as follows.

Lemma 4. Preventing a in aw

Suppose $w = b^{x_1}ab^{x_2}a \ldots ab^{x_n}$ is solvable side-place-freely. Then in aw, all occurrences of a can be separated side-place-freely.

Proof: In order to prevent a in w side-place-freely at any state r_k, the system (6) has a solution with $a_+ = 0$ and $b_- = 0$ for any fixed $1 \leq k \leq n-1$. This refers to a pure input place q of a, which may or may not be an output place of b. In order to prevent a in aw side-place-freely, we need to consider the states r_k as before (but shifted to the right by one index position, still just before the last b of the k'th group of b's) and a correspondingly modified system as follows:

$$\begin{array}{ll} 0 \leq m' + (x_1 + \ldots + x_i) \cdot (b'_+) + (i+1) \cdot (-a'_-) & \text{for all } 0 \leq i \leq n-1 \\ 0 \leq -m' - (x_1 + \ldots + x_k - 1) \cdot (b'_+) - k \cdot (-a'_-) + a'_- - 1 & \end{array} \tag{8}$$

where m', b'_+ and a'_- refer to a new pure place q' preventing a at state r_k in aw. The line with $i = 0$ was added because m' is required to be bounded from below and a must be enabled initially. (In (6), nonnegativity of m follows from the line with $i = 1$ and b being the first transition of w, which is no longer true in aw.) Consider the transformation

$$m' = m + a_- \text{ and } b'_+ = b_+ \text{ and } a'_- = a_-$$

These values satisfy (8), provided m, b_+ and a_- (together with $a_+ = 0$ and $b_- = 0$) satisfy (6). The line with $i = 0$ follows from $m' = m + a_- \geq 0$. The other lines corresponding to $i \geq 1$ reduce to the corresponding lines in (6), since the additional $(-a_-)$ at the end of each line is offset by the additional $(+a_-)$ at the beginning of the line. The last line (which belongs to state r_k at which a is separated) corresponds to the last line of (6), because the decrease by a_- at the beginning of the line is offset by an increase by a_- in the term $k \cdot (-a'_-)$ (compared with $(k-1) \cdot (-a_-)$ as in (6)). □ 4

Note 1: In order to disable a at r_k, q could be replaced by a place q' obtained by duplicating q and changing the initial marking m to $m' = m + a_-$. Intuitively, this means that m' is computed from m by "unfiring" a once.

Note 2: Place q should not be removed as soon as q' is added, because q could also be preventing a at some other r_k. In that case, a new place q'' must be computed from q for this different value of k. We may forget about q only after all the relevant indices k have been processed.

Next, consider an input place p of b in a side-place-free solution of w and suppose that p prevents b at state s_k. Suppose that we want to solve aw. If p is not also an output place of a, then it can simply be retained unchanged, and with the same marking, prevent b at corresponding states in aw and in w. However, if p is also an output place of a, there may be a problem, because "unfiring" a in the initial marking may lead to negative tokens on p. This is illustrated by the example *babbabb* which has a side-place-free solution, as shown on the left-hand side of figure 5.

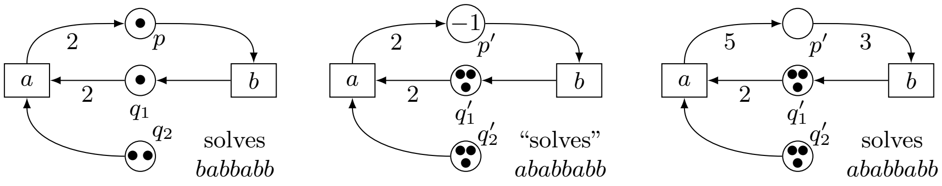


Fig. 5. Solving *babbabb* (l.h.s.), (almost) *ababbabb* (middle), and *ababbabb* (r.h.s.).

The places q_1, q_2 can be treated as in the above proof (that is, by changing their markings by "unfiring" a) and yield new places q_1', q_2' with marking $\{(q_1', 3), (q_2', 3)\}$. If we allowed negative markings, then a new place p' with initial marking $(p', -1)$ (and otherwise duplicating p) would do the job of solving *ababbabb* (as in the middle of the figure). However, we shall need a more refined argument in order to avoid negative markings.

Let p' be a general new place which is supposed to prevent b at state s_k in aw. In order to check the general solvability of aw if w is side-place-freely solvable, we consider a general transformation

$$m' = m+\mu \;\;,\;\; b_+' = b_+ + \beta_+ \;\;,\;\; b_-' = b_- + \beta_- \;\;,\;\; a_+' = a_+ + \alpha_+ \;\;,\;\; a_-' = a_- + \alpha_-$$

where $\mu \geq -m$, $\beta_+ \geq -b_+$, $\beta_- \geq -b_-$, $\alpha_+ \geq -a_+$ and $\alpha_- \geq -a_-$, as well as a new inequation system:

$$\begin{array}{ll} b_+' & \leq m' + (x_1 + \ldots + x_i) \cdot (b_+' - b_-') + i \cdot (a_+' - a_-') \quad \text{for } 1 \leq i \leq n \\ 0 & \leq -m' - (x_1 + \ldots + x_k) \cdot (b_+' - b_-') - k \cdot (a_+' - a_-') + b_-' - 1 \end{array}$$

This system has to be compared with a restricted form of (5) (setting $b_+ = a_- = 0$, since the solution of w is pure). Doing this by line-wise comparison, we get the following inequation system for the new value differences:

$$\begin{array}{lll} \mu & \geq & -m,\ \ \beta_+ \ \geq\ -b_+,\ \ \beta_- \ \geq\ -b_-,\ \ \alpha_+ \ \geq\ -a_+,\ \ \alpha_- \ \geq\ -a_- \\ \beta_+ & \leq & \mu + (x_1 + \ldots + x_i) \cdot (\beta_+ - \beta_-) + i \cdot (\alpha_+ - \alpha_-) + a_+ \\ 0 & \leq & -\mu - (x_1 + \ldots + x_k) \cdot (\beta_+ - \beta_-) - k \cdot (\alpha_+ - \alpha_-) - a_+ + \beta_- \end{array} \tag{9}$$

The lines with i must be solved simultaneously for every $1 \leq i \leq n$ while the line with k must be solved individually for every $1 \leq k \leq n-1$, in order to get a place preventing b at state s_k. This leads to the following lemma.

Lemma 5. Solving aw from w

Suppose $w = b^{x_1}ab^{x_2}a \ldots ab^{x_n}$ is solvable side-place-freely. Then aw is solvable.

Proof: Suppose that a pure place p with parameters b_- (arc into b), a_+ (arc from a) and m (initial marking) is given and suppose it separates b from s_k in w. This place solves (5) for that particular k. We distinguish two cases:

Case 1: $a_+ \leq m$. In this case, the place p can essentially be re-used for the same purpose in the solution (that we construct in this way) for aw, since (9) is solved by putting

$$\mu = -a_+ \ ,\ \beta_+ = \beta_- = 0 \ ,\ \alpha_+ = \alpha_- = 0$$

Hence, a place p' which differs from p only by its initial marking ($m' = m - a_+$ instead of m) separates b at s_k in aw.

Case 2: $a_+ > m$. In this case, (9) can be solved by

$$\mu = -m \ ,\ \beta_+ = \beta_- = a_+ - m \ ,\ \alpha_+ = \alpha_- = 0$$

That is, we may replace p by a place p' with zero initial marking and adding uniformly the value $a_+ - m$ to the incoming and outgoing arcs of b, creating a side-place around b. □ 5

For instance, in the solution of *babbabb* shown on the left-hand side of figure 5, the place p from a to b satisfies m=1, b_-=1, b_+=0, a_-=0 and a_+=2. (9) is solved by $\mu = -1$, $\beta_- = 2$, $\beta_+ = 0$, $\alpha_- = 0$ and $\alpha_+ = 3$. Hence with m'=$m-1$, b'_-=$b_- + 2$, b'_+=b_+, a'_-=a_- and a'_+=$a_+ + 3$, the net shown on the right-hand side of figure 5 is a pure solution of *ababbabb*. (Place p' prevents b not only in states s_1 and s_2 but also in the initial state and in the final state.) There exist words such as $w_1 = bbbabab$ or $w_2 = bbabbababab$, however, which can be solved but for which aw is not solvable. We have a converse of Lemma 5:

Lemma 6. Solving w side-place-freely from aw

If aw has a solution, then w has a side-place-free solution.

Proof: Suppose that aw has a solution in which some place q', preventing a, has a side-place around a. Because q' prevents a, $a'_- > a'_+$ (unless it is the first a, but then we don't need q' in solving w). Because a is enabled initially, $m' \geq a'_-$. But then, the transformation $a''_- = a'_- - a'_+$, $a''_+ = 0$, $m'' = m' - a'_+$ yields another place q'' which is not a side-place around a but serves the same purpose as q'. The rest of the proof follows because the above transformations (removing side-places around b, or side-places around a which prevent b) do not introduce any new side-places around a. □ 6

Corollary 2. Side-place-free solvability of $b^{x_1}ab^{x_2}a\ldots ab^{x_n}$

$w = b^{x_1}ab^{x_2}a\ldots ab^{x_n}$ *is solvable side-place-freely iff* aw *is solvable.* □ 2

6 Concluding remarks

If the characterisations of minimal Petri net solvable binary words proposed in this paper are valid, then the usual linear solver for detecting them can be replaced by a pattern-matching algorithm based on conjecture 1, or by a letter-counting algorithm based on conjecture 2. We have described a variety of results providing some insight into this class of words. There are several other facts about them which we did not have space to describe. For example, if a word is solvable side-place-freely, then so is the reverse word. Also, if a word is solvable, then it is solvable by places having exactly one outgoing transition. (This property is not shared by words with three or more letters, a counterexample being *abcbaa*.) Moreover, PN-solvable words are balanced in the following sense. Referring to $w = b^{x_1}ab^{x_2}a\ldots ab^{x_n}$, call w balanced if there is some x such that $x_i \in \{x, x+1\}$ for all $2 \leq i \leq n-1$. We can prove that if $w = b^{x_1}ab^{x_2}a\ldots ab^{x_n}$ is PN-solvable, then w is balanced, and moreover, $x_n \leq x+1$. Presenting these, and other, properties of PN-solvability must however be left to future publications.

Acknowledgements: We would like to thank Harro Wimmel and Uli Schlachter for valuable comments, and Uli for finding weird words (amongst them, *abcbaa* as mentioned in the conclusion) using [7, 3]. We thank also the anonymous reviewers for their remarks which allowed to improve the presentation of the paper.

References

1. É. Badouel, P. Darondeau: Theory of Regions. In W. Reisig, G. Rozenberg (eds): Lectures on Petri Nets I: Basic Models. LNCS Vol. 1491, 529–586 (1998).
2. E. Best, R. Devillers: State Space Axioms for T-Systems. G. Lüttgen, F. Corradini (eds): Special volume on the occasion of Walter Vogler's 60th birthday. Acta Informatica, Vol. 52(2-3), 133-152 (2015).
3. E. Best, U. Schlachter: Analysis of Petri Nets and Transition Systems. ICE'2015.
4. B. Caillaud: `http://www.irisa.fr/s4/tools/synet/`
5. T. Murata: Petri Nets: Properties, Analysis and Applications. Proc. of the IEEE, Vol. 77(4), 541-580 (1989).
6. M. Piątkowski et al.: `http://folco.mat.umk.pl/unsolvable-words` (2015).
7. U. Schlachter et al.: `https://github.com/CvO-Theory/apt` (2013).

Event Log Visualisation with Conditional Partial Order Graphs: from Control Flow to Data

Andrey Mokhov[1], Josep Carmona[2]

[1] Newcastle University, United Kingdom
andrey.mokhov@ncl.ac.uk
[2] Universitat Politècnica de Catalunya, Spain
jcarmona@cs.upc.edu

Abstract Process mining techniques rely on *event logs*: the extraction of a process model (*discovery*) takes an event log as the input, the adequacy of a process model (*conformance*) is checked against an event log, and the *enhancement* of a process model is performed by using available data in the log. Several notations and formalisms for event log representation have been proposed in the recent years to enable efficient algorithms for the aforementioned process mining problems. In this paper we show how *Conditional Partial Order Graphs (CPOGs)*, a recently introduced formalism for compact representation of families of partial orders, can be used in the process mining field, in particular for addressing the problem of compact and easy-to-comprehend visualisation of event logs with data. We present algorithms for extracting both the control flow as well as the relevant data parameters from a given event log and show how CPOGs can be used for efficient and effective visualisation of the obtained results. We demonstrate that the resulting representation can be used to reveal the hidden interplay between the control and data flows of a process, thereby opening way for new process mining techniques capable of exploiting this interplay.

1 Introduction

Event logs are ubiquitous sources of process information that enabled the rise of the *process mining* field, which stands at the interface between data science, formal methods, concurrency theory, machine learning, data visualisation and others [26]. A *process* is a central notion in these fields and in computing science in general, and the ability to automatically discover and analyse evidence-based process models is of utmost importance for many government and business organisations. Furthermore, this ability is gradually becoming a necessity as the digital revolution marches forward and traditional process analysis techniques based on the explicit construction of precise process models are no longer adequate for continuously evolving large scale real-life processes, because our understanding of them is often incomplete and/or inconsistent.

At present, the process mining field is mainly focused on three research directions: i) the *discovery* of a formal process model, typically, a Petri Net or a BPMN (Business Process Model and Notation); ii) the *conformance* analysis of a process model with respect to a given event log; and iii) the *enhancement* of a process model with respect to additional information (i.e., *data*) contained in an event log. The bulk of research in these directions has been dedicated to the design of the algorithmic foundation and associated software tools with many notable successes, such as, e.g. the ProM framework [2]. However, a more basic problem of *event log visualisation* received little attention to date, despite the fact that effective visualisation is essential for achieving a good understanding of the information contained in an event log. Indeed, even basic *dotted charts* prove very useful for describing many aspects of event logs even though they are just simple views of event log traces plotted over time [25].

In this paper we discuss the application of *Conditional Partial Order Graphs (CPOGs)* for event log visualisation. The CPOG model has been introduced in [17] as a compact graph-based representation for complex concurrent systems, whose behaviour could be thought of as a collection of many partial order scenarios. The key idea behind our approach is to convert a given event log into a collection of partial orders, which can then be compactly described and visualised as a CPOG. Although CPOGs are less expressive than Petri Nets and have important limitations, such as the inability to represent cyclic behaviour, they are perfectly suitable for representing event logs, which are inherently acyclic. We therefore see CPOGs not as the final product of process mining, but as a convenient intermediate representation of an event log that provides much better clarity of visualisation as well as better compactness, which is important for the efficiency of algorithms further in the process mining pipeline. Furthermore, CPOGs can be manipulated using algorithmically efficient operations such as *overlay* (combining several event logs into one), *projection* (extracting a subset of interesting traces from an event log), *equivalence checking* (verifying if two event logs describe the same behaviour) and others, as has been formalised in [19].

The contribution of the paper is twofold. Firstly, we present a method for deriving compact CPOG representations of event logs, which is based on the previous research in CPOG *synthesis* [17]. Secondly, we propose techniques for extracting data parameters from the information typically contained in *event labels* of a log and for using these parameters for annotating the derived CPOG model, thereby providing a direct link between the control and data aspects of a given system.

The remainder of the paper is organised as follows: the next section illustrates the motivation and contributions of the paper with the help of a small example. Section 3 provides the background on event logs, and Section 4 introduces the theory of CPOGs in detail, placing it in the context of process mining. The extraction of CPOGs from event logs is described in Section 5. This is followed by Section 6, which shows how one can automatically incorporate data into CPOGs. Finally, Section 7 provides a discussion about related and future work.

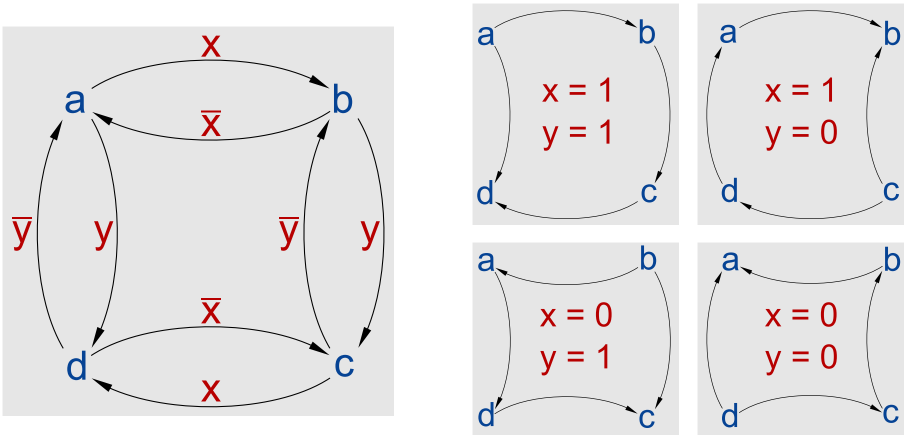
2 Motivating Example

We start by illustrating the reasons that motivate us to study the application of CPOGs in process mining, namely: (i) the ability of CPOGs to compactly represent complex event logs and clearly illustrate their high-level properties, and (ii) the possibility of capturing event log meta data as part of a CPOG representation, thereby taking advantage of the meta data for the purpose of explaining the process under observation.

Consider an event log $L = \{abcd, cdab, badc, dcba\}$. One can notice that the order between events a and b always coincides with the order between events c and d. This is an important piece of information about the process, which however may not be immediately obvious when looking at the log in the text form. To visualise the log one may attempt to use existing process mining techniques and discover a graphical representation for the log, for example in the form of a Petri Net or a BPMN. However, the existing process mining techniques perform very poorly on this log and fail to capture this information. To compare the models discovered from this log by several popular process mining methods, we will describe the discovered behaviour by regular expressions, where operators $||$ and $\cup$ denote interleaving and union, respectively.

The α-algorithm [27] applied to L produces a Petri Net accepting the behaviour $a \cup b \cup c \cup d$, which clearly cannot reproduce any of the traces in L. Methods aimed at deriving block-structured process models [3][13] produce a connected Petri Net that with the help of *silent* transitions reproduces the behaviour $a\,||\,b\,||\,c\,||\,d$, which is a very imprecise model accepting all possible interleavings of the four events. The region-based techniques [4] discover the same behaviour as the block-structured miners, but the derived models are not connected.

CPOGs, however, can represent L exactly and in a very compact form, as shown in Fig. 1(a). Informally, a CPOG is an overlay of several partial orders that can be extracted from it by assigning values to variables that appear in the conditions of the CPOG vertices and arcs, e.g., the upper-left graph shown in Fig. 1(b) (assignment $x = 1$, $y = 1$) corresponds to the partial order containing the causalities $a \prec b$, $a \prec d$, $b \prec c$, $c \prec d$. One can easily verify that the model is precise by trying all possible assignments of variables x and y and checking that they generate the traces $\{abcd, cdab, badc, dcba\}$ as expected, and nothing else. See Fig. 1(b) for the corresponding illustration. The compactness of the CPOG representation of L is due to the fact that several event orderings can be overlayed on top of each other taking advantage of the similarities between them. See Sections 4 and 5 for a detailed introduction to CPOGs and algorithms for automated translation of event logs to CPOGs.



(a) CPOG representation

(b) Four projections of the CPOG

Figure 1: Exact CPOG representation of log $L = \{abcd, cdab, badc, dcba\}$

It is worth mentioning that CPOGs allow us to recognise *second order relations* between events. These are relations that are not relating events themselves, but are relating relations between events: indeed, the CPOG in Fig. 1(a) clearly shows that the relation between a and b is equal to the relation between c and d, and the same holds for pairs (a, d) and (b, c). In principle, one can go even further and consider third order relations and so forth. The practical use of such a relation hierarchy is that it may help to extract an event hierarchy from event logs, thereby simplifying the resulting representation even further.

One may be unsatisfied by the CPOG representation in Fig. 1(a) due to the use of 'artificial' variables x and y. Where do these variables come from and what exactly do they correspond to in the process? We found out that additional data which is often present in event logs can be used to answer such questions. In fact, as we will show in Section 6, it may be possible to use easy-to-understand predicates constructed from the data instead of 'opaque' Boolean variables.

For example, consider the same log L but augmented with temperature data attached to traces:

- $abcd, t = 25^\circ$
- $cdab, t = 30^\circ$
- $badc, t = 22^\circ$
- $dcba, t = 23^\circ$

With this information at hand we can now explain what variable x means. In other words, we can open the previously opaque variable x by expressing it as a predicate involving data parameter t:

$$x = t \geq 25^\circ$$

One can subsequently drop x completely from the CPOG by using conditions $t \geq 25^\circ$ and $t < 25^\circ$ in place of x and $\overline{x}$, respectively. See Fig. 2 for the corresponding illustration.

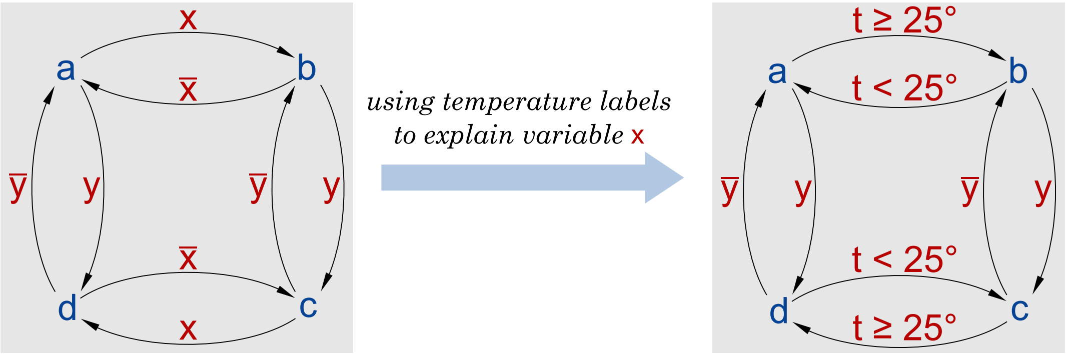


Figure 2: Using data to explain variables

To conclude, we believe that CPOGs bring unique log visualisation capabilities to the process mining field. It is possible to use CPOGs as an intermediate representation of event logs, which can be exact as well as more comprehensible both for humans and for software tools further in the process mining pipeline.

3 Event Logs

In this section we introduce the notion of an *event log*, which is central for this paper and for the process mining field. We also discuss important quality metrics that are typically used to compare methods for event log based process mining.

Table 1 shows a simple event log, which contains not only event information but also *data* in the form of *event attributes*. The example event log matches the log used in the previous section,

Event	Case ID	Activity	Timestamp	Temperature	Resource	Cost	Risk
1	1	a	10-04-2015 9:08am	25.0	Martin	17	Low
2	2	c	10-04-2015 10:03am	28.7	Mike	29	Low
3	2	d	10-04-2015 11:32am	29.8	Mylos	16	Medium
4	1	b	10-04-2015 2:01pm	25.5	Silvia	15	Low
5	1	c	10-04-2015 7:06pm	25.7	George	14	Low
6	1	d	10-04-2015 9:08pm	25.3	Peter	17	Medium
7	2	a	10-04-2015 10:28pm	30.0	George	19	Low
8	2	b	10-04-2015 10:40pm	29.5	Peter	22	Low
9	3	b	11-04-2015 9:08am	22.5	Mike	31	High
10	4	d	11-04-2015 10:03am	22.0	Mylos	33	High
11	4	c	11-04-2015 11:32am	23.2	Martin	35	High
12	3	a	11-04-2015 2:01pm	23.5	Silvia	40	Medium
13	3	d	11-04-2015 7:06pm	28.8	Mike	43	High
14	3	c	11-04-2015 9:08pm	22.9	Silvia	45	Medium
15	4	b	11-04-2015 10:28pm	23.0	Silvia	50	High
16	4	a	11-04-2015 10:40pm	23.1	Peter	35	Medium

Table 1: An example event log

that is the underlying traces are $\{abcd, cdab, badc, dcba\}$ and they correspond to 'case IDs' 1, 2, 3, and 4, respectively. We assume that the set of attributes is fixed and the function *attr* maps pairs of events and attributes to the corresponding values. For each *event* e the log contains the case ID $case(e)$, the activity name $act(e)$, and the set of attributes defined for e, e.g., $attr(e, timestamp)$. For instance, for the event log in Table 1, $case(e_7) = 2$, $act(e_7) = a$, $attr(e_7, timestamp) =$ "10-04-2015 10:28pm", and $attr(e_7, cost) = 19$. Given a set of events E, an *event log* $L \in \mathbb{B}(E^*)$ is a multiset of *traces* E^* of events.

Process mining techniques use *event logs* containing footprints of real process executions for discovering, analysing and extending formal process models, which reveal real processes in a system [26]. The process mining field has risen around a decade ago, and since then it has evolved in several directions, with process discovery being perhaps the most difficult challenge, as demonstrated by the large number of techniques available for it today. What makes process discovery hard is the fact that derived process models are expected to be good across four quality metrics, which are often mutually exclusive:

- *fitness*: the ability of the model to reproduce the traces in the event log (i.e., not too many traces are lost),
- *precision*: the precision of the model in representing the behavior in the log (i.e., not too many new traces are introduced),
- *generalisation*: the ability of the model to generalise the behavior not covered by the log, and
- *simplicity*: the well-known *Occam's Razor* principle that advocates for simpler models.

Although this paper does not focus on the discovery of process models, we will consider these quality metrics when analysing the derived Conditional Partial Order Graphs, which are formally described in the next section.

4 Conditional Partial Order Graphs

Conditional Partial Order Graphs (CPOGs) were introduced for the compact specification of concurrent systems comprised from multiple behavioural scenarios [17]. CPOGs are particularly effective when scenarios of the system share common patterns, which can be exploited for the automated derivation of a compact combined representation of the system's behaviour. CPOGs have been used for the design of asynchronous circuits [20] and for optimal encoding of processor instructions [18]. In this paper we demonstrate how CPOGs can be employed in process mining.

4.1 Basic definitions

A CPOG is a directed graph (V, E), whose *vertices* V and *arcs* $E \subseteq V \times V$ are labelled with Boolean functions, or *conditions*, $\phi : V \cup E \rightarrow (\{0,1\}^X \rightarrow \{0,1\})$, where $\{0,1\}^X \rightarrow \{0,1\}$ stands for a Boolean function defined on Boolean *variables* X.

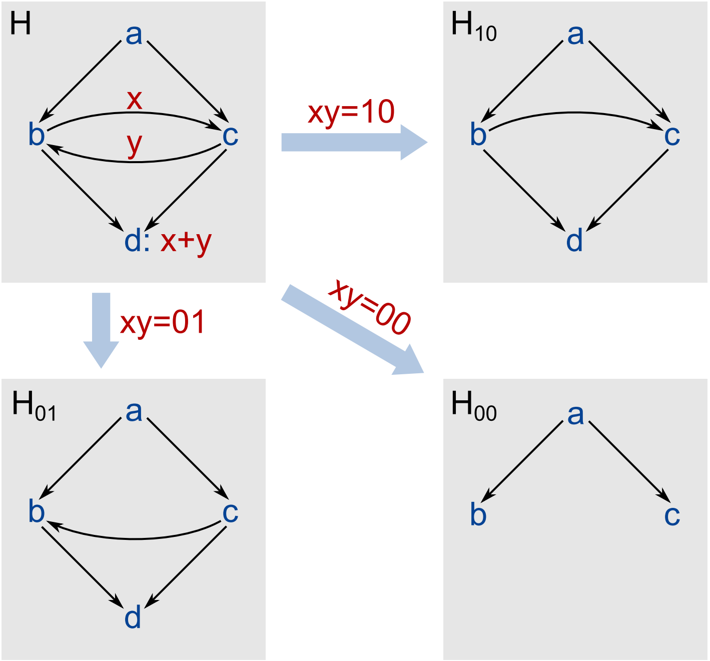


Figure 3: Example of a Conditional Partial Order Graph and the associated family of graphs

Fig. 3 (the top left box) shows an example of a CPOG H containing 4 vertices $V = \{a, b, c, d\}$, 6 arcs and 2 variables $X = \{x, y\}$. Vertex d is labelled with condition $x + y$ (that is, 'x OR y'), arcs (b, c) and (c, b) are labelled with conditions x and y, respectively. All other vertices and arcs are labelled with trivial conditions 1 (trivial conditions are not shown for clarity); we call such vertices and arcs *unconditional*.

There are $2^{|X|}$ possible assignments of variables X, called *codes*. Each code induces a subgraph of the CPOG, whereby all the vertices and arcs, whose conditions evaluate to 0 are removed. For example, by assigning $x = y = 0$ one obtains graph H_{00} shown in the bottom right box in Fig. 3; vertex d and arcs (b, c) and (c, b) have been removed from the graph, because their conditions are equal to 0 when $x = y = 0$. Different codes can produce different graphs, therefore a CPOG with $|X|$ variables can potentially specify a *family* of $2^{|X|}$ graphs. Fig. 3 shows two other members of the family specified by CPOG H: H_{01} and H_{10}, corresponding to codes 01 and 10, respectively, which differ only in the direction of the arc between vertices b and c.

It is often useful to focus only on a subset $C \subseteq \{0, 1\}^X$ of codes, which are meaningful in some sense. For example, code 11 applied to CPOG H in Fig. 3 produces a graph with a loop between vertices b and c, which is undesirable if arcs are interpreted as causality. We use a Boolean *restriction function* $\rho : \{0, 1\}^X \to \{0, 1\}$ to compactly specify the set $C = \{\mathbf{x} \mid \rho(\mathbf{x}) = 1\}$ and its complement $DC = \{\mathbf{x} \mid \rho(\mathbf{x}) = 0\}$, which are often referred to as the *care* and *don't care* sets [8]. By setting $\rho = \overline{xy}$ one can disallow code $\mathbf{x} = 11$, thereby restricting the family of graphs specified by CPOG H to three members only, which are all shown in Fig. 3.

The *size* $|H|$ of a CPOG $H = (V, E, X, \phi, \rho)$ is defined as:

$$|H| = |V| + |E| + |X| + \left| \bigcup_{z \in V \cup E} \phi(z) \cup \rho \right|,$$

where $|\{f_1, f_2, \ldots, f_n\}|$ stands for the size of the smallest circuit [30] that computes all Boolean functions in set $\{f_1, f_2, \ldots, f_n\}$.

4.2 Families of partial orders

A CPOG $H = (V, E, X, \phi, \rho)$ is *well-formed* if every allowed code $\mathbf{x}$ produces an acyclic graph $H_\mathbf{x}$. By computing the transitive closure $H^*_\mathbf{x}$ one can obtain a *strict partial order*, an irreflexive and transitive relation on the set of *events* corresponding to vertices of $H_\mathbf{x}$.

We can therefore interpret a well-formed CPOG as a specification of a family of partial orders. We use the term *family* instead of the more general term *set* to emphasise the fact that partial orders are *encoded*, that is each partial order $H^*_\mathbf{x}$ is paired with the corresponding code $\mathbf{x}$. For example, the CPOG shown in Fig. 3 specifies the family comprising the partial order H^*_{00}, where event a precedes concurrent events b and c, and two total orders H^*_{01} and H^*_{10} corresponding to sequences *acbd* and *abcd*, respectively.

The *language* $\mathcal{L}(H)$ *of a CPOG* H is the set of all possible linearisations of partial orders contained in it. For example, the language of the CPOG shown in Fig. 3 is $\mathcal{L}(H) = \{abc, acb, abcd, acbd\}$. One of the limitations of the CPOG model is that it can only describe finite languages. However, this limitation is irrelevant for the purposes of this paper since event logs are always finite.

It has been demonstrated in [14] that CPOGs are a very efficient model for representing families of partial orders. In particular, they can be exponentially more compact than *Labelled Event Structures* [21] and *Petri Net unfoldings* [15]. Furthermore, for some applications CPOGs provide more comprehensible models than other widely used formalisms, such as *Finite State Machines* and *Petri Nets*, as has been shown in [17] and [20]. This motivated the authors to investigate the applicability of CPOGs to process mining.

4.3 Synthesis

In the previous sections we have demonstrated how one can extract partial orders from a given CPOG. However, the opposite problem is more interesting: *derive the smallest CPOG description* *for a given a set of partial orders*. This problem is called *CPOG synthesis* and it is an essential step in the proposed CPOG-based approach to process mining.

A number of CPOG synthesis methods have been proposed to date. In this paper we will rely on the one based on graph colouring [17], which produces CPOGs with all conditions having at most one literal. Having at most one literal per condition is a serious limitation for many applications, but we found that the method works well for process mining. A more sophisticated approach, which produces CPOGs with more complex conditions has been proposed in [18], however, it has poor scalability and cannot be applied to large process mining instances. Both methods are implemented in open-source Workcraft framework [1], which we used in our experiments.

In general, the CPOG synthesis problem is still under active development and new approximate methods are currently being studied, e.g., see [7]. Another promising direction for overcoming this challenge is based on reducing the CPOG synthesis problem to the problem of Finite State Machine synthesis [29].

5 From Event Logs to CPOGs

When visualising behaviour of an event log, it is difficult to identify a single technique that performs well for any given log due to the *representational bias* exhibited by existing process discovery algorithms. For example, if the event log describes a simple workflow behaviour, then the α-algorithm [27] is usually the best choice. However, if non-local dependencies are present in the behaviour, the α-algorithm will not be able to find them, and then other approaches, e.g. based on the theory of regions [4][24][28], may deliver best results. The latter techniques in turn are not tailored for dealing with noise, and alternative approaches such as [9][31] should be considered. There are event logs for which none of the existing process discovery techniques seem to provide a satisfactory result according to the quality metrics presented in Section 3; see the simple event log shown in Section 2 as an example.

In this section we describe two approaches for translating a given event log L into a compact CPOG representation H. The first approach, which we call the *exact CPOG mining*, treats each trace as a totally ordered sequence of events and produces CPOG H such that $L = \mathcal{L}(H)$. The second approach attempts to extract concurrency between the events, hence we call it the *concurrency-aware CPOG mining*. The former approach does not introduce any new behaviours, while the latter one may in fact introduce new behaviours, which could be interpreted as new possible interleavings of the traces contained in the given log, hence producing CPOG H such that $L \subseteq \mathcal{L}(H)$.

5.1 Exact CPOG mining

The problem of the *exact CPOG mining* is formulated as follows: given an event log L, derive a CPOG H such that $L = \mathcal{L}(H)$. This can be trivially reduced to the CPOG synthesis problem. Indeed, each trace $t = e_1 e_2 \cdots e_m$ can be considered a total order of events $e_1 \prec e_2 \prec \cdots \prec e_m$. Therefore, a log $L = \{t_1, t_2, \cdots, t_n\}$ can be considered a set of n total orders and its CPOG representation can be readily obtained via CPOG synthesis. Note that the solution always exists, although it is not unique. If uniqueness is desirable one can fix the assignment of codes to traces, in which case the result of synthesis can be presented in so-called *canonical form* [19].

For example, given event log $L = \{abcd, cdab, badc, dcba\}$ described in Section 2, the exact mining approach produces the CPOG shown in Fig. 1. As has already been discussed in Section 2, the resulting CPOG is very compact and provides a more comprehensible representation of the event log compared to conventional models used in process mining, such as Petri Nets or BPMNs.

When a given event log contains concurrency, the exact CPOG mining approach may lead to suboptimal results. For example, consider a simple event log $L = \{abcd, acbd\}$. If we directly synthesise a CPOG by considering each trace of this log a total order, we will obtain the CPOG H shown in Fig. 4 (left). Although $L = \mathcal{L}(H)$ as required, the CPOG uses a redundant variable x to distinguish between the two total orders even though they are just two possible linearisations of the same partial order, where $a \prec b$, $a \prec c$, $b \prec d$, and $c \prec d$. It is therefore desirable to recognise and extract the concurrency between events b and c in this event log and use the information for simplifying the derived CPOG, as shown in Fig. 4 (right). Note that the simplified CPOG H' still preserves the language equality, i.e. $L = \mathcal{L}(H')$.

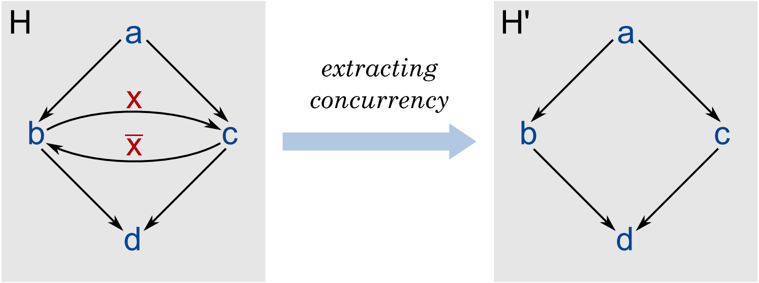


Figure 4: CPOG mining from event log $L = \{abcd, acbd\}$

5.2 Concurrency-aware CPOG mining

This section presents an algorithm for extracting concurrency from a given event log and using this information for simplifying the result of the CPOG mining. Classic process mining techniques generally follow the same principle; in particular, the α-algorithm [26] is often used to extract concurrency in the context of process mining based on Petri Nets. We introduce a new concurrency extraction algorithm, which is more conservative than the α-algorithm: it uses stronger restrictions when declaring two events concurrent, which leads to higher accuracy of process mining. This method works particularly well in combination with CPOGs due to their compactness, however, we believe that it can also be useful in combination with other formalisms.

First, let us introduce convenient operations for extracting subsets of traces from a given event log L. Given an event e, the subset of L's traces containing e will be denoted as $L|_e$, while the subset of L's traces not containing e will be denoted as $L|_{\overline{e}}$. Clearly, $L|_e \cup L|_{\overline{e}} = L$. Similarly, given two events e and f, the subset of L's traces containing both e and f with e occurring before f will be denoted as $L|_{e\rightarrow f}$. Note that $L|_e \cap L|_f = L|_{e\rightarrow f} \cup L|_{f\rightarrow e}$, i.e., if two events appear in a trace, they must be ordered one way or another. For instance, if $L = \{abcd, acbd, abce\}$ then $L_e = \{abce\}$, $L_{\overline{a}} = \emptyset$, $L_{a\rightarrow b} = L$, and $L_{a\rightarrow d} = \{abcd, acbd\}$. An event e is *conditional* if $L|_e \neq \emptyset$ and $L|_e \neq L$, otherwise it is *unconditional*. A conditional event will necessarily have a non-trivial condition (neither 0 nor 1) in the mined CPOG. Similarly, a pair of events e and f is *conditionally ordered* if $L|_{e\rightarrow f} \neq \emptyset$ and $L|_{e\rightarrow f} \neq L$. Otherwise, e and f are *unconditionally ordered.*

We say that a conditional event r *indicates* the order between events e and f in an event log L if one of the following criteria holds:

- $L|_r \subseteq L|_{e \to f}$
- $L|_r \subseteq L|_{f \to e}$
- $L|_{\overline{r}} \subseteq L|_{e \to f}$
- $L|_{\overline{r}} \subseteq L|_{f \to e}$

In other words, the existence or non-existence of the event r can be used as an indicator of the order between the events e and f. For example, if $L = \{abcd, acbd, abce\}$, then e indicates the order between b and c. Indeed, whenever we observe event e in a trace we can be sure that b occurs before c in that trace: $L|_e \subseteq L|_{b \to c}$.

Similarly, we say that a conditionally ordered pair of events r and s *indicates* the order between events e and f in an event log L if one of the following criteria holds:

- $L|_{r \to s} \subseteq L|_{e \to f}$
- $L|_{r \to s} \subseteq L|_{f \to e}$
- $L|_{s \to r} \subseteq L|_{e \to f}$
- $L|_{s \to r} \subseteq L|_{f \to e}$

In other words, the order between the events r and s can be used as an indicator of the order between the events e and f. For example, if $L = \{abcd, cdab, badc, dcba\}$, then the order between events a and b indicates the order between events c and d (and vice versa). Indeed, whenever a occurs before b in a trace, we know that c occurs before d: $L|_{a \to b} = L|_{c \to d}$.

The *indicates relation* has been inspired by and is somewhat similar to the *reveals relation* introduced in [10].

We are now equipped to describe the algorithm for concurrency-aware CPOG mining. The algorithm takes an event log L as input and produces a CPOG H such that $L \subseteq \mathcal{L}(H)$.

1. Extract concurrency: find all conditionally ordered pairs of events e and f such that the order between them is not indicated by any other events or pairs of events. Call the resulting set of pairs C.
2. Convert each trace $t \in L$ into a partial order p by relaxing the corresponding total order according to the set of concurrent pairs C. Call the resulting set of partial orders P.
3. Perform the CPOG synthesis on the obtained set of partial orders P to produce the resulting CPOG H.

Note that the resulting CPOG H indeed satisfies the condition $L \subseteq \mathcal{L}(H)$, since we can only add new linearisations into H in step (2) of the algorithm, when we relax a total order corresponding to a particular trace by discarding some of the order relations.

Let us apply the algorithm to the previous examples. Given log $L = \{abcd, cdab, badc, dcba\}$ from Section 2, the algorithm does not find any concurrent pairs, because the order between each pair of events is indicated by the order between the complementary pair of events (e.g., $L|_{a \to b} = L|_{c \to d}$). Hence, $C = \emptyset$ and the result of the algorithm coincides with the exact CPOG mining, as shown in Section 2. Given log $L = \{abcd, acbd\}$ from Section 5.1, the algorithm finds one pair of concurrent events, namely $\{b, c\}$, which results in collapsing of both traces of L into the same partial order with trivial CPOG representation shown in Fig. 4 (right).

6 From Control Flow to Data

As demonstrated in the previous section, one can derive a compact CPOG representation from a given event log using CPOG mining techniques. The obtained representations however rely on opaque Boolean variables, which make the result difficult to comprehend. For example, Fig. 1(a) provides no intuition on how a particular variable assignment can be interpreted with respect to the process under observation. The goal of this section is to present a method for the automated extraction of useful data labels from a given event log (in particular from available event attributes) and using these labels for constructing 'transparent' and easy-to-comprehend predicates, which can substitute the opaque Boolean variables. This is similar to the application of conventional machine learning techniques for learning 'decision points' in process models or in general for the automated enhancement of a given model by leveraging the available data present in the event log [26].

More formally, given an event log L and the corresponding mined CPOG H our goal is to explain how a particular condition f can be interpreted using data available in the log L. Note that the condition f can be as simple as just a single literal $x \in X$ (e.g., the arc $a \to b$ in Fig. 1(a)), in which case our goal is to explain a particular Boolean variable; however, the technique introduced in this section is applicable to any Boolean function of the CPOG variables $f : \{0,1\}^X \to \{0,1\}$, in particular, one can use the technique for explaining what the restriction function ρ corresponds to in the process, effectively discovering the process *invariants*. We achieve the goal by constructing an appropriate instance of the *classification problem* [16].

Let $n = |E|$ be the number of different events in L, and k be the number of different event attributes available in L. Remember that attributes of an event e can be accessed via function $attr(e)$, see Section 3. Hence, every event e in the log defines a *feature vector* $\hat{e}$ of dimension k where the value at i-th position corresponds to the value of the i-th attribute of e[3]. For instance, the feature vector $\hat{e}_1$ corresponding to the event e_1 in the log shown in Table 1 is ("10-04-2015 9:08am", 25.0, "Martin", 17, Low). Some of the features, e.g. timestamp, may need to be abstracted before applying the technique described below in order to produce better results. For example, timestamps can be mapped to five discrete classes *morning*, *noon*, *afternoon*, *evening* and *night*.

The key observation for the proposed method is that all traces in the log L can be split into two disjoint sets, or *classes*, with respect to the given function f: i) set $L|_f$, containing the traces where f evaluates to 1, and ii) set $L|_{\overline{f}}$ containing the traces where f evaluates to 0. This immediately leads to an instance of the *binary classification problem* on n feature vectors, as illustrated in Table 2.

Feature vectors	Class
$\{\hat{e} \mid e \in \sigma \wedge \sigma \in L|_f\}$	True
$\{\hat{e} \mid e \in \sigma \wedge \sigma \in L|_{\overline{f}}\}$	False

Table 2: Binary classification problem for function f and event log L.

In other words, every event belonging to a trace where the function evaluates to 1 is considered to belong to the class we learn, that is, the class labelled as True in Table 2 (the remaining events do not belong to this class). Several methods can be applied to solve this problem, including *decision*

[3] We assume a total order on the set of event attributes.

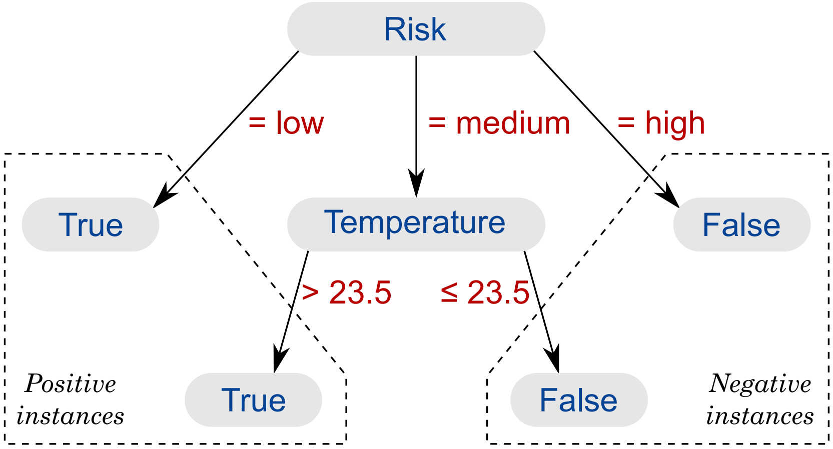


Figure 5: Decision tree built for function $f = x$ in the CPOG of Fig. 1(a).

trees [23], *support vector machines* [6], and others. In this work we focus on decision trees as they provide a convenient way to extract predicates defined on event attributes, which can be directly used for substituting opaque CPOG conditions. The method is best explained by way of an example.

Consider the event log shown in Table 1, which contains a number of data attributes for each event. The traces underlying the log are $\{abcd, cdab, badc, dcba\}$. Fig. 1(a) shows the corresponding CPOG produced by the CPOG mining techniques presented in the previous section. Let us try to find an interpretation of the variable x by applying the above procedure with $f = x$. The set $L|_f$ equals to $L|_{a\rightarrow b}$, i.e. it contains traces 1 and 2, wherein event a occurs before event b and therefore $f = 1$. Therefore, feature vectors $\hat{e}_1$-$\hat{e}_8$ provide the positive instances of the class to learn (the first eight events of the log belong to traces 1 and 2), while feature vectors $\hat{e}_9$-$\hat{e}_{16}$ provide the negative ones. The decision tree shown in Fig. 5 is a possible classifier for this function, which has been derived automatically using machine learning software Weka [11]. By combining the paths in the tree that lead to positively classified instances, one can derive the following predicate for f: $\mathit{risk} = \mathit{low} \vee (\mathit{risk} = \mathit{medium} \wedge \mathit{temperature} > 23.5)$. This predicate can be used to substitute the opaque variable x in the mined CPOG.

One can use the same procedure for deriving the explanation for all variables and/or conditions in the mined CPOG, thereby providing a much more comprehensible representation for the event log. Note that for complementary functions, taking the negation of the classification description will suffice, e.g., conditions $\overline{x}$ in Fig. 1(a) can be substituted with predicate $\mathit{risk} \neq \mathit{low} \wedge (\mathit{risk} \neq \mathit{medium} \vee \mathit{temperature} \leq 23.5)$. Alternatively, one can derive the predicate for a complementary function by combining paths leading to the negative instances; for example, for $f = \overline{x}$ the resulting predicate is $\mathit{risk} = \mathit{high} \vee (\mathit{risk} = \mathit{medium} \wedge \mathit{temperature} \leq 23.5)$.

The learned classifier can be tested for evaluating the quality of representation of the learned concept. If the quality is unacceptable then the corresponding condition may be left unexplained in the CPOG. Therefore in general the data extraction procedure may lead to partial results when the process contains concepts which are 'difficult to learn'. For example, in the discussed case study the condition $f = y$ could not be classified exactly.

A coarse-grain alternative to the technique discussed in this section is to focus on *case attributes* instead of event attributes. Case attributes are attributes associated with a case (i.e., a trace) as a whole instead to individual events [26]. Furthermore, the two approaches can be combined with the aim of improving the quality of obtained classifiers.

7 Discussion

The techniques presented in this paper are currently being implemented as part of the Workcraft framework [1][22], and the next step is to evaluate them on real-life event logs containing data attributes. Several challenges need to be faced, e.g., the complexity of the concurrency extraction algorithm (the first step in the algorithm presented in Section 5.2), the fine-tuning of parameters of the machine learning techniques, and some others.

Due to the inability of CPOGs to directly represent cyclic behavior, we currently only focus on using CPOGs for visualisation and as an intermediate representation of event logs, which can be further transformed into an appropriate process mining formalism, such as Petri Nets or BPMNs. Although some syntactic transformations already exist to transform CPOGs into contextual Petri nets [22], we believe that finding new methods for discovery of process mining models from CPOGs is an interesting direction for future research.

Another research direction is to consider CPOGs as compact algebraic objects that can be used to efficiently manipulate and compare event logs [19]. Since a CPOG corresponding to an event log can be exponentially smaller, this may help to alleviate the memory requirements bottleneck for current process mining tools that store ‘unpacked’ event logs in memory.

Event logs are not the only suitable input for the techniques presented in this paper: we see an interesting link with the work on *discovery of frequent episodes*, e.g., as reported recently in [12]. *Episodes* are partially ordered collections of events (not activities), and as such they can also be represented by CPOGs. This may help to compress the information provided by frequent episodes, especially if one takes into account the fact that current algorithms may extract a large number of episodes, which then need to be visualised for human understanding.

8 Conclusions

This paper describes the first steps towards the use of CPOGs in the field of process mining. In particular, the paper presented the automatic derivation of the control flow part of the CPOG representation from a given event log, and then the incorporation of meta data contained in the log as conditions of the CPOG vertices and arcs. We have implemented some of the reported techniques, in particular the extraction of a CPOG from an event log as described in Section 5, and some preliminary experiments have been carried out.

The future work includes addressing the challenges described in the previous section, as well as a thorough practical evaluation of the algorithms described in this paper. The developed software tool may then be used within a more general framework such as ProM [2], Workcraft [1] or PMLAB [5].

Acknowledgments. The authors would like to thank various organisations that supported this research work. Andrey Mokhov was supported by Royal Society Research Grant ‘Computation Alive’ and EPSRC project UNCOVER (EP/K001698/1). Josep Carmona was partially supported by funds from the Spanish Ministry for Economy and Competitiveness (MINECO) and the European Union (FEDER funds) under grant COMMAS (ref. TIN2013-46181-C2-1-R).

References

1. The Workcraft framework homepage. http://www.workcraft.org/, 2009.
2. The ProM framework homepage. http://www.promtools.org/, 2010.
3. Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. A genetic algorithm for discovering process trees. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2012, Brisbane, Australia, June 10-15, 2012*, pages 1–8, 2012.
4. Josep Carmona, Jordi Cortadella, and Michael Kishinevsky. New region-based algorithms for deriving bounded Petri nets. *IEEE Trans. Computers*, 59(3):371–384, 2010.
5. Josep Carmona and Marc Solé. PMLAB: an scripting environment for process mining. In *Proceedings of the BPM Demo Sessions 2014 Co-located with the 12th International Conference on Business Process Management (BPM 2014), Eindhoven, The Netherlands, September 10, 2014.*, pages 16 – 21, 2014.
6. Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.
7. A. de Gennaro, P. Stankaitis, and A. Mokhov. A heuristic algorithm for deriving compact models of processor instruction sets. In *International Conference on Application of Concurrency to System Design (ACSD)*, 2015.
8. G. de Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education, 1994.
9. Christian W. Günther and Wil M. P. van der Aalst. Fuzzy mining - adaptive process simplification based on multi-perspective metrics. In Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors, *BPM*, volume 4714 of *Lecture Notes in Computer Science*, pages 328–343. Springer, 2007.
10. Stefan Haar, Christian Kern, and Stefan Schwoon. Computing the reveals relation in occurrence nets. *Theor. Comput. Sci.*, 493:66–79, 2013.
11. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.
12. Maikel Leemans and Wil M. P. van der Aalst. Discovery of frequent episodes in event logs. In *Proceedings of the 4th International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2014), Milan, Italy, November 19-21, 2014.*, pages 31–45, 2014.
13. Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Discovering block-structured process models from event logs - A constructive approach. In *Application and Theory of Petri Nets and Concurrency - 34th International Conference, PETRI NETS 2013, Milan, Italy, June 24-28, 2013. Proceedings*, pages 311–329, 2013.
14. H. Ponce De Leon and A. Mokhov. Building bridges between sets of partial orders. In *International Conference on Language and Automata Theory and Applications (LATA)*, 2015.
15. KL McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *Proceedings of Computer Aided Verification conference (CAV)*, volume 663, page 164, 1992.
16. Tom M. Mitchell. *Machine learning*. McGraw Hill series in computer science. McGraw-Hill, 1997.
17. A. Mokhov. *Conditional Partial Order Graphs*. PhD thesis, Newcastle University, 2009.
18. A. Mokhov, A. Alekseyev, and A. Yakovlev. Encoding of processor instruction sets with explicit concurrency control. *Computers & Digital Techniques, IET*, 5(6):427–439, 2011.
19. A. Mokhov and V. Khomenko. Algebra of Parameterised Graphs. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(4s):143, 2014.
20. A. Mokhov and A. Yakovlev. Conditional Partial Order Graphs: Model, Synthesis, and Application. *IEEE Transactions on Computers*, 59(11):1480–1493, 2010.
21. M. Nielsen, G. D. Plotkin, and G. Winskel. Petri nets, event structures and domains, part I. *Theoretical Computer Science*, 13:85–108, 1981.
22. Ivan Poliakov, Danil Sokolov, and Andrey Mokhov. Workcraft: a static data flow structure editing, visualisation and analysis tool. In *Petri Nets and Other Models of Concurrency–ICATPN 2007*, pages 505–514. Springer, 2007.
23. J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

24. Marc Solé and Josep Carmona. Light region-based techniques for process discovery. *Fundam. Inform.*, 113(3-4):343–376, 2011.
25. Minseok Song and Wil MP van der Aalst. Supporting process mining by showing events at a glance. *Proceedings of the 17th Annual Workshop on Information Technologies and Systems (WITS)*, pages 139–145, 2007.
26. Wil van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.
27. Wil M. P. van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE TKDE*, 16(9):1128–1142, 2004.
28. Jan Martijn E. M. van der Werf, Boudewijn F. van Dongen, Cor A. J. Hurkens, and Alexander Serebrenik. Process discovery using integer linear programming. In *ATPN*, pages 368–387, 2008.
29. Tiziano Villa, Timothy Kam, Robert K Brayton, and Alberto L Sangiovanni-Vincentelli. *Synthesis of finite state machines: logic optimization.* Springer Publishing Company, Incorporated, 2012.
30. Ingo Wegener. *The Complexity of Boolean Functions.* Johann Wolfgang Goethe-Universitat, 1987.
31. A.J.M.M. Weijters, W.M.P. van der Aalst, and A.K. Alves de Medeiros. Process mining with the heuristics miner-algorithm. Technical Report WP 166, BETA Working Paper Series, Eindhoven University of Technology, 2006.

Discovery of Personal Processes from Labeled Sensor Data – An Application of Process Mining to Personalized Health Care

Timo Sztyler[1], Johanna Völker[1], Josep Carmona[2],
Oliver Meier[1], Heiner Stuckenschmidt[1]

[1]University of Mannheim, Germany
{timo,johanna,heiner}@informatik.uni-mannheim.de
[2]Universitat Politècnica de Catalunya, Spain
jcarmona@cs.upc.edu

Abstract. Currently, there is a trend to promote personalized health care in order to prevent diseases or to have a healthier life. Using current devices such as smart-phones and smart-watches, an individual can easily record detailed data from her daily life. Yet, this data has been mainly used for *self-tracking* in order to enable personalized health care. In this paper, we provide ideas on how process mining can be used as a fine-grained evolution of traditional self-tracking. We have applied the ideas of the paper on recorded data from a set of individuals, and present interesting conclusions and challenges.

1 Introduction

Physical inactivity is a major risk factor for certain types of diseases. Indeed, physical activity does not only prevent or relieve diseases, but also improves public health and well being [2]. In this context, personalized health solutions and lifestyle monitoring can help to ensure that people doing the right activity at the right time. However, the regular use of such methods is critical to achieve the desired result. Hence, barriers for the adoption must be low, and using both software and devices should be as comfortable as possible.

Thanks to the technological progress in the development of wearable devices, sensor technology, and communication, we are nowadays able to setup a body sensor network based on smart-phones, smart-watches, and wristbands which does not affect people during their daily routine. In contrast, most of the available software requires substantial user input to specify, e.g., the current activity or even vital parameters like the heart rate or blood pressure.

We want to develop an application which monitors the personal lifestyle of the users and provides appropriate visualizations. However, this still needs a sufficient acceptance because the user has to view and interpret the visualizations. Therefore, we also want to provide automatically generated recommendations resulting from the monitoring data and, e.g., references (practical guidance). In the long term, we also have to automatically recognize a person's daily activities such as different types of sports and desk work. This is necessary to ensure that the required user input is a minimum which also is a requirement to make the application practical.

Due to the fact that the activities of a user can be seen as process instances, process mining can help us to elicit and analyze these processes. It allows discovering a process model from an event log focused on personal activity, and combined with, e.g., conformance checking, to explore deviations with respect to reference models. The results could be useful in the context of monitoring to provide a meaningful feedback but also to create recommendations.

In this paper, we present the data set we created for our first experiments (see Section 2), and we outline initial ideas about how process mining could help us to address our main use cases (see Sections 3, 4, and 5):

Monitoring We want to help users to monitor their personal behavior by presenting them a daily or weekly visual summary of their personal processes. This summary could highlight behavior which is unknown or unconscious to the user. As a result, the user could correct the behavior.

Deviations We want compare their personal processes with reference processes to detect deviations. This allows making suggestions regarding the procedure of certain activities, and point out missing activities. As a result, the user learns to optimize the daily routine in respect of a healthy lifestyle.

Operational Support Historical data that combines both activity and environmental data (e.g., geographic position) can then be used for the operational support based on individual's process models, enabling predictions and recommendations in order to accomplish certain goals.

We do not deal with activity recognition but address succeeding problems. The created data set is a training data set with manually labeled data. Commonly, machine learning techniques are used for activity recognition [9]. Therefore, the data set can be used to build or evaluate activity recognition systems, but in the following we want to use the result of such a system in combination with process mining to create personal processes by using the manually created activity labels. The resulting personal process models should allow to benefit the users health by making visualizations, recommendations, and predictions.

2 Data Gathering

This section provides the details of the data set used in this paper. The data set can be obtained by contacting us.

General Settings. Seven individuals (age 23.1±1.81, height 179.0±9.09, weight 80.6±9.41, seven males) collected Accelerometer, Orientation, and GPS sensor data and labeled this data simultaneously (see Table 1). The data was collected using a smart-phone and smart-watch combined with a self-developed sensor data collector and labeling framework (see Figure 1). The subjects were not supervised but got an introduction and guidelines. The subject group covers five students, a worker, and a researcher.

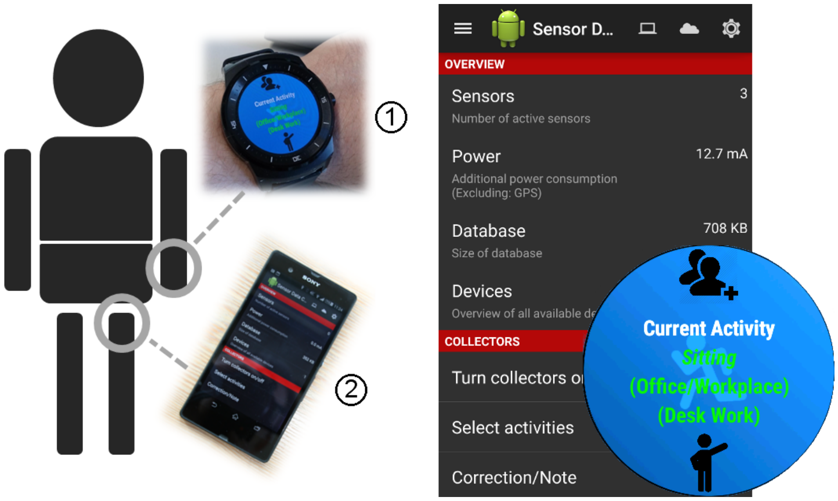


Fig. 1. Collector and labeling framework: Wear App (smart-watch, 1) and Hand App (smart-phone, 2). The positions of the devices may vary.

Setup	
Subjects	7 males
Devices	Smart-phone (2) and Smart-watch (1)
Sensors	Acceleration (50Hz), Orientation (50Hz), GPS (every 10 min.)
Labels	Activity, Device Position, Environment, Posture
Storage	Local Database, SD-Card
Duration	10 hours a day, 12 days

Table 1. Equipment and Settings of the data gathering.

Devices and Labeling. The framework consists of a *Wear* (1) and *Hand* (2) application which interact with each other via Bluetooth. The *Wear* application allows updating the parameters (see Table 1) immediately where the *Hand* application manages the settings of the sensors and the storing of the data. The sampling rate (50Hz) was chosen with consideration of battery life as well as with reference to previous studies [12, 19]. Table 1 summaries the equipment and settings.

The individuals should collect data during their daily routine and it was up to them to decide where the device should be positioned on the body. We focused on the activity, device position, environment, and the posture which occur during the daily routine. The values for these parameter were predefined (see Tables 2 and 3) and could not be changed or extended.

Activities. The activity labels allow recording the daily routine. We focused on food intake, sport, different type of movements, but also (house) work so that we can compare the daily routine of several individuals to detect common activity

Parameter	**Values**
Device Position	Chest, Hand, Head, Hip, Forearm, Shin, Thigh, Upper Arm, Waist
Environment	Building, Home, Office, Street, Transportation
Posture	Climbing, Jumping, Lay, Running, Sitting, Standing, Walking

Table 2. Labeling parameters that were updated immediately when the *device position*, *environment*, or *posture* had changed.

Activity	**Sub-Activity**
Desk Work[1]	*n/a*
Eating/Drinking	Breakfast, Brunch, Coffee Break, Dinner, Lunch, Snack
Housework	Cleaning, Tidying Up
Meal Preparation[1]	*n/a*
Movement	Go for a Walk, Go Home, Go to Work
Personal Grooming[1]	*n/a*
Relaxing	Playing, Listen to Music, Watching TV
Shopping[1]	*n/a*
Socializing	Bar/Disco, Cinema at Home
Sleeping[1]	*n/a*
Sport	Basketball, Bicycling, Dancing, Gym, Gymnastics, Ice Hockey, Jogging, Soccer
Transportation	Bicycle, Bus, Car, Motorcycle, Scooter, Skateboard, Train, Tram

Table 3. Activity and sub-activity labels. The subjects had to select at least one of these activity labels to specify their current action. The selection of a sub-activity is optional but allows to be more precise. [1]Please note, that there are activities without sub-activities.

patterns but also to analyze the different behaviors. The set of activity labels was minimized and structured to decrease the time which the individual needs to decide and choose a suitable label. Thus, there are 12 activities and 32 sub-activities where an activity could be "Eating/Drinking" and a corresponding sub-activity "Breakfast"[1]. It is possible to select several activity labels at the same time to record the current situation with a high accuracy (e.g., "Movement - go to Work", "Transportation - Train", and "Sleeping"). Thus, the individual can describe the current situation from several points of view.

To keep the set of activity labels as small as possible, we provided some generic labels such as "Desk Work". This label should be used if the individual works in an office (worker), attends a lecture or class room (student), or visits a school (pupil). During the introduction phase, we explained this to the individuals to avoid that they choose different labels in the same situation.

[1] Please note: So far, we do not consider the sub-activities in the presented use-cases.

	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15	D16	D17	D18	D19	D20	D21	D22	D22	D23
S1											■	■	■	■		■	■	■	■	■				
S2								■	■	■	■	■			■	■	■	■	■					
S3		■	■			■	■	■	■	■		■	■		■	■		■						
S4	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■									
S5		■	■	■	■		■		■	■	■		■	■	■	■	■	■						
S6		■		■																				
S7									■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■

Table 4. Timetable. Overview of how many and which days where recorded for each individual. The X-axis represents the [D]ays whereas the Y-axis illustrates the [S]ubjects. The grey colored day labels (D[0-9]+) are weekend days. The grey squares indicate that data was recorded.

Profiling. We recorded 74 cases which cover 1,386 events. A case is represented by one individual in one particular day and has an average duration of 12.1 hours. The events comprise the activities and sub-activities which were performed by the individuals. Table 4 describes when and how long each individual recorded data. Tables 5 and 6 illustrate the related recorded data. The number of records of acceleration and orientation differs, because one subject selected a lower frequency for the orientation sensor. The high standard deviation of the numbers of postures results from the different behavior of the individuals. Hence, some individuals move a lot (e.g., walking, standing, walking) while others label the posture less accurate (e.g., standing just for a second).

Labels	**Records** (avg±sd)
Activities	20 ± 7
Postures	80 ± 62
Environment	16 ± 4
Dev. Position	8 ± 6

Table 5. Annotated labels per day and individual.

Raw Data	**Records** (absolute)
Acceleration	$2.7 * 10^6$
Orientation	$2.3 * 10^6$
geo. Location	70

Table 6. Number of recorded values per day and individual.

3 Use Case 1: Monitor Personal Behavior

Since a picture is worth a thousand words, the deployment of graphical representations of event data may open the door to a precise awareness of the activities carried out by an individual. We believe graphs are a strong visualization aid to understand aggregated behavior, and thus consider this direction as the first use case for understanding personal activity data.

Interesting information a user can get periodically (every day or week) is the personal process model that describes the main activities and their dependencies.

In this process model, one can find frequent sequences of activities, alternatives, concurrency (moving while eating) and so on. This deviates from the typical information that is provided by current tools for self-tracking individuals. In general, such tools focus only on showing correlations between the tracked variables (e.g., eating vs. sport) or the evolution of single variables (weight over the week). In this section, we take the training data that were described in the previous section and illustrate how traditional process discovery techniques can be used to elicit the personal process model of an individual. The preliminary conclusions reported in this section should not be considered as a general rule but instead are meant to illustrate the capabilities of process discovery techniques in providing a fresh look for self-tracking.

3.1 Focusing on the Frequent Paths

Due to the variability in personal activity data, there is not a simple process model that represents all possible paths for an individual, even for the reduced number of individuals monitored in this paper. In this section, we focus on the most frequent paths taken by each individual. To this end, the discovery of *fuzzy models* [8] using the Disco tool [4] is considered. The reason for using a frequency-based discovery technique is to handle the variability and noise of a self-tracking log. Alternative techniques like the *heuristic miner* [18] or the *inductive miner* [10] which can be applied in this scenario may be considered as well.

To illustrate the potential of a personal process model with respect to analyzing tons of raw data, we focus on two simple aspects: the difference in activity between work and weekend days on the one hand, and the differences across the individuals on the other hand.

During the week vs. weekend. Figures A.1 and A.2 (see Appendix), show the main activity models during the working week and the weekend, respectively. The process models depicted in the figures have a very different structure. This clearly denotes a variation in the personal activity during the week and weekend, when considering the main activity by individuals. For instance, while in the week days the main behavior is centered towards "Desk Work" which is also the most frequent activity, the frequency of paths and activities is more balanced in the weekends. This tendency is also satisfied in the average duration of activities (not shown in the process models).

Personal activity across users. Figure A.3 (see Appendix) shows each individual's main activity models. As it was explained in the previous section, three types of individuals were monitored: student (5 instances), researcher (1 instance) and worker (1 instance). Although the details of the models are not visible in this figure, one can see significant differences across individuals. Commonalities between students are also elicited in the models, for example, the global tendency to structure the model around "Desk Work" and the well-structured relation between the activities for most of the students.

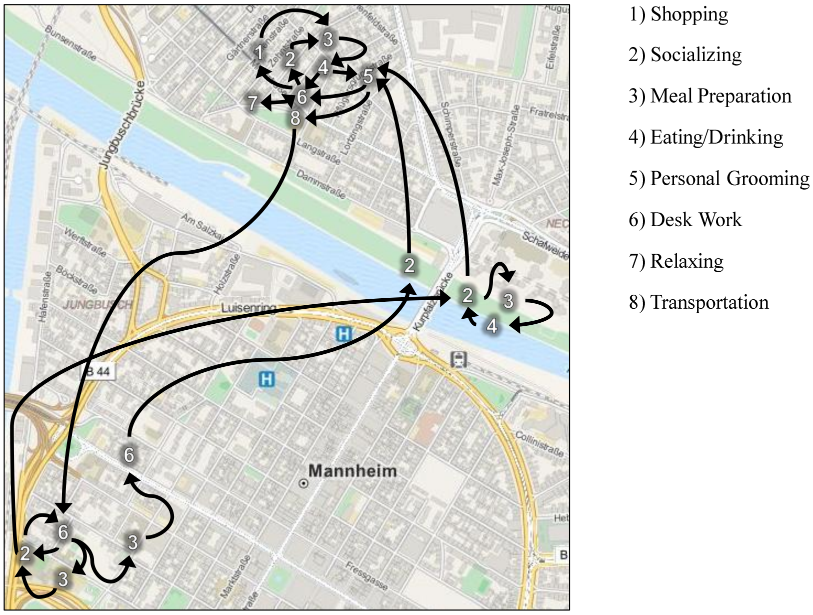


Fig. 2. Main personal activity for an individual including geographical position data: numbers correspond to different activities, and arcs denote control-flow relations extracted from the activity data.

3.2 Model Enhancement Using Personal Data

As shown in Section 2, not only activity data is stored from individuals but also important data like the geographical position, acceleration and, orientation of the device. In the following, as an example, we explain how to combine the control-flow process models (e.g., see Figure A.1) with the geographical position data to derive *personal activity-position maps*. This kind of map illustrates geographically the control-flow with respect to the real geographical position of activities. Figure 2 depicts an example of such a map for the data gathered from one of the individuals. The computation of personal activity-position maps can be done by simply aligning the timing information (*start*, *end*) recorded for each activity event with the one obtained from the geographical position of individuals. This way, for every activity, its geographical position in a case will be extracted. Events corresponding to the activity name will be then analyzed to compute a set of locations that represents the different locations where the activity has been carried out. For instance, in Figure 2, activity 2 ("Socializing") has four different nodes in the graph. Ideally, to have a simpler graph, only one location per activity is desired. The locations for an activity can be computed by clustering the set of locations with a fixed radius of k meters and selecting the centroids, or by using the frequency of locations, or a combination of both. Finally, the nodes corresponding to each activity in a certain location are displayed on top of a real map, the area of which corresponds to the minimal enclosing box that includes all locations depicted. Arcs from the control-flow are then routed from the corresponding locations in the map.

The personal activity-position maps are strongly related to trajectory pattern mining [20]. A trajectory pattern consists of chronologically ordered geographical locations combined with the duration. The provided algorithms allow to detect frequent behaviors in space and time (daily, weekly), and in this context to aggregate movement behavior of a person [7] or a group [14] [11] to keep track on specific movements. This facilitates to discover highly frequented places as well as underlying patterns in movements which might be related to other persons, and can help to identify semantic relations between persons [11]. Related to this, a previous work [13] explored the principle limitations of predicting human dynamics based on mobility patterns of smart-phone users.

Concerning our scenario, we focus on the daily routine of a person and the related activities which means that we have to connect the spatiotemporal information explicitly with the activity information. If we can combine this information and apply the mentioned techniques then it could help to influence the daily routine of a person in terms of achieving a healthier life by optimizing specific kind of patterns. Considering health care, the kind of transportation between locations might be also important in the context of energy consumption but this is not covered by the mentioned techniques (see [7]).

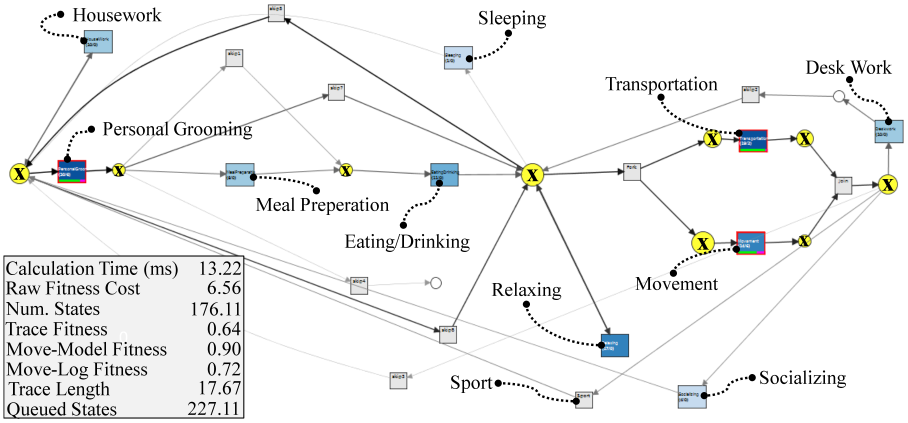
4 Use Case 2: Deviations from Reference Models

Self-tracking may be a meaningful way to verify if certain requirements with respect to reference quantities are accomplished. For instance, many associations advise to do at least 30 minutes of moderate physical activity per day or eat fish at least twice a week. Those guidelines for a good lifestyle offer a rough description for individuals, mainly concerning about quantities and frequencies. However, some ways of satisfying these guidelines are probably less healthy than others, e.g., it may not be the best decision to eat fish while doing physical activity.

Hence, there may be reference models that describe precisely how activities should be carried out in order to satisfy a guideline. Thus, the reference model has to provide the opportunity to describe certain actions in a specific *order* (e.g., “Sport” should be followed by “Personal Grooming”), should allow explicit *choices* (e.g., after “Desk Work” only “Eating/Drinking”, “Socializing”, or “Transportation” are expected actions) and should also consider *concurrency* actions. (e.g., “Transportation” and “Movement” may be overlapping activities).

Reference models can be obtained in several ways. One possibility would be to ask a domain expert to create manually the desired reference model for a given goal. A second option would be to collect event logs from successful individuals. These logs can be combined with the introduced techniques of the previous section to discover a reference model. Finally, a third option would be to translate the

[3] `http://www.promtools.org`



Calculation Time (ms)	13.22
Raw Fitness Cost	6.56
Num. States	176.11
Trace Fitness	0.64
Move-Model Fitness	0.90
Move-Log Fitness	0.72
Trace Length	17.67
Queued States	227.11

Fig. 3. Example of fitness analysis in ProM[3] of an individual with respect to a reference model: places with yellow background (X) represent situations where the individual deviates from the process model. Transitions without a label denote silent events not appearing in the event log.

textual guidelines into process models, using recent techniques that apply *natural language processing* to elicit process models [6].

When a reference model is available, *conformance checking* techniques can be applied to assess the adequacy of the reference process model in representing the traces of individuals [15]. Since the reference model describes the ideal behavior, it is meaningful to focus the analysis on the *fitness* of the reference model with respect to the traces of individuals. A process model fits a given trace if it can reproduce it. An example of such analysis can be seen in Figure 3 where an individual is analyzed with respect to an invented process model meant to represent a healthy behavior.

Fitness checking can also be extended to consider other perspectives, i.e., costs or quantities for additional event data [5]. For instance, one typical advice on dietary guidelines is *to eat as many calories as one burns* [1]. These kind of checks can be incorporated into the reference model by using the data conformance approach from [5]. Therefore, deviations on quantities can also be verified with respect to the reference model.

If reference models are not available, simple rules can be used which should be satisfy by individuals on their daily routine. These rules may describe patterns that should satisfy an individual, e.g., "taking medicines" should be followed by "eating". This can be formally specified with Linear Temporal Logic (LTL) formulas to be satisfied by the event log of activities [17].

5 Use Case 3: Operational Support

Historical data of an individual is a rich source of information which may be crucial to influence the daily routine in order to reach a particular goal. In this context, process models can be enhanced and used at each decision point to assess the influence of the next step in satisfying the targeted goal. For instance, following the guideline of the previous section that advice to eat as many calories as one burns, activities can be annotated with respect to calorie levels (e.g., "Eating/Drinking" produces an amount of calories while "Movement" takes an amount of calories). Then, historical activity data can be aggregated with this information to learn for all decision points the impact of the decision regarding the likelihood of satisfying the targeted goal, e.g., the balanced consumption of calories. Figure 4 shows an example for the case of the balance of calories in a diet, i.e., states (nodes) are labeled with the probability of reaching a balanced diet at the end of the day.

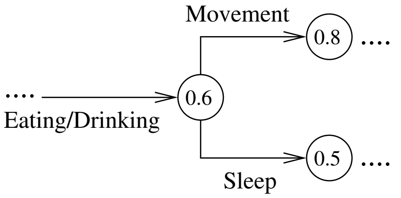


Fig. 4. Excerpt of a state-based prediction model for balance of calories. The nodes illustrate the probability for reaching the balance.

Thus, when an individual is about to start a new activity, recommendations can be provided on the basis on the model's aggregated data corresponding to the current state. This deviates from current prediction and recommendation practices that do not consider the current state of the model explicitly.

The precision of the prediction may vary due to the fact that the available information can have a different granularity. Hence, events can carry information such as the amount of calories but also only cover complete cases with the resulting label (e.g., good, satisfactory, medium, bad). In such a case, standard techniques [15] for the operational support of process models can be applied to predict and recommend the next steps.

6 Future Work

In the following, we outline a few general directions of future work and possible next steps.

When process mining is applied, e.g., to identify and visualize the most frequent paths, it should take into account a given hierarchy of activities and subactivities. Such a hierarchy could facilitate, for instance, the aggregation of collected data on different levels of abstraction.

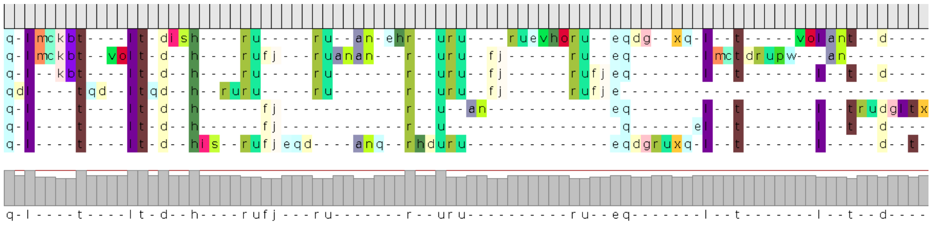

q - l - - - - t - - - - l t - d - - h - - - - r u f j - - - r u - - - - - - - r - - u r u - - - - - - - - - - r u - - e q - - - - - - - l - - t - - - - - - - l - - t - - d - - - -

Fig. 5. Example of discovered trace cluster: letters in the bottom denote activities with high consensus. The Y-axis represents seven different traces where the X-axis illustrates the different events per traces.

Future applications of process mining might also require dealing with uncertain data. In particular, the data generated by classification-based methods for activity recognition will most probably be uncertain, since these methods are never a hundred percent accurate. However, provenance information such as explicit uncertain values will be available in most cases, and might serve as an additional input to process mining methods.
Further directions include the investigation of more expressive process models. For example, reference models, which describe an ideal sequence of daily routines, should include information about frequencies, time and locations.
Finally, we would like to bootstrap activity recognition by creating and leveraging synergies between activity recognition and process mining techniques. A possible bootstrapping approach would generate process models from automatically recognized activities, and use the resulting process models to improve the accuracy of the activity recognition.

More concrete ideas that we would like to investigate are the following:

Exploring the log via trace alignment. Section 3.1 focused on the main activity paths followed by individuals, thus ignoring less frequent behavior that may mislead the conclusions. An alternative will be to preprocess the log with the goal of extracting patterns, and then transform the log accordingly, either by introducing hierarchy, or by ignoring outlier activities not following the learned patterns. For this purpose, *Trace alignment* techniques from [3] can be applied. For instance, in Figure 5 seven traces have been aligned together from the log of workdays.

Process Cubes. Recently, *process cubes* have been proposed also as a means to apply process mining in a exploratory manner, similar to *online analytical processing* (OLAP) techniques [16]. The intuitive idea is to mine event logs by restricting events under a particular perspective. For example, extracting a process model for activity in a bank, focusing only on clients from a given region that got married within the last three years. With the data available in a personal activity context, process cubes can be a promising way to slice the data and mine particular contexts. For instance, one can be interested in process models where “Desk Work” is mainly situated in a given location.

7 Conclusions

This paper discusses challenges and opportunities for process mining in the area of personalized health care. We described the acquisition of a real-world data set consisting of manually labeled sensor data from smart-phones, and outlined interesting use cases. We then took a look at existing methods for eliciting, analyzing and monitoring individuals' daily routines, and described the results of our preliminary experiments. We presented our ideas on future directions and challenges in this application context which may require significant advances with respect to algorithmic support for process mining.

Acknowledgments. This work as been partially supported by funds from the Spanish Ministry for Economy and Competitiveness (MINECO) and the European Union (FEDER funds) under grant COMMAS (ref. TIN2013-46181-C2-1-R).

A Appendix

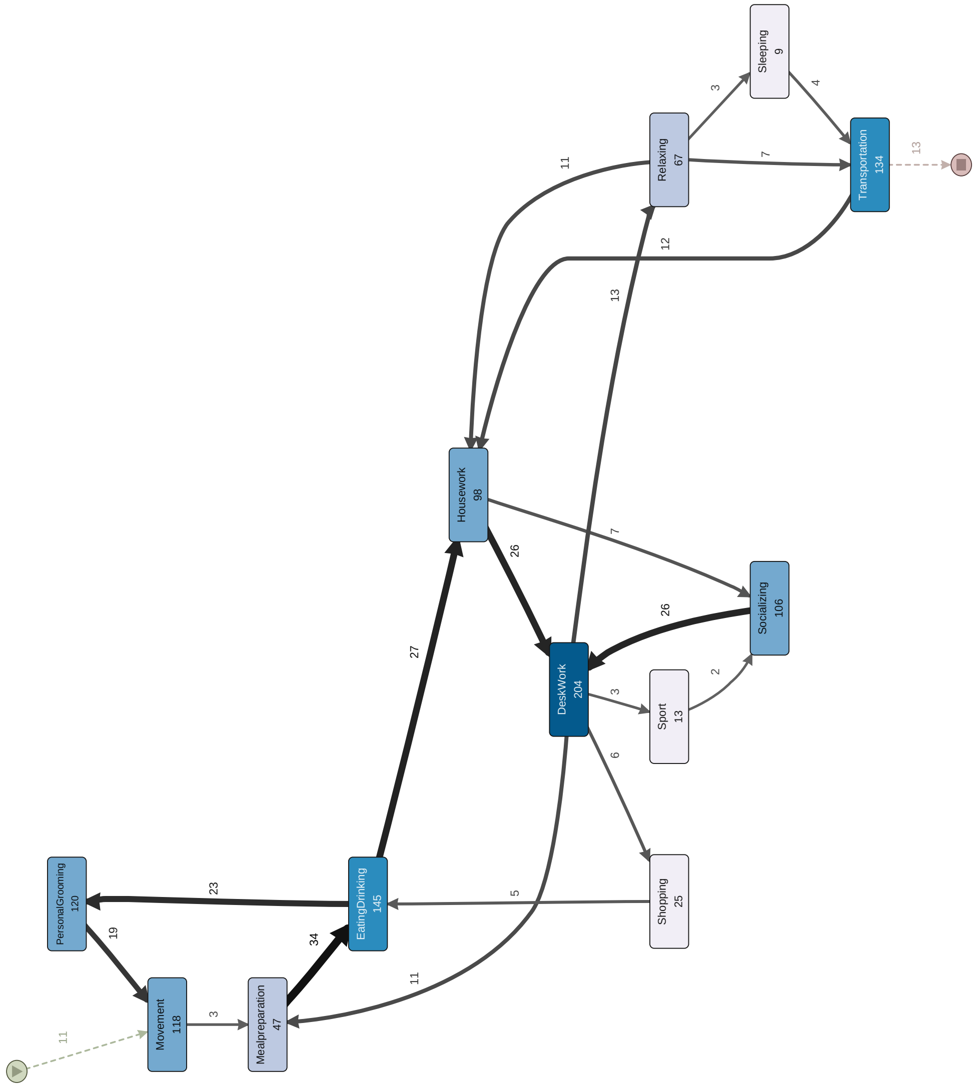


Fig. A.1. Main personal activity for all the users during the working week days (57 cases).

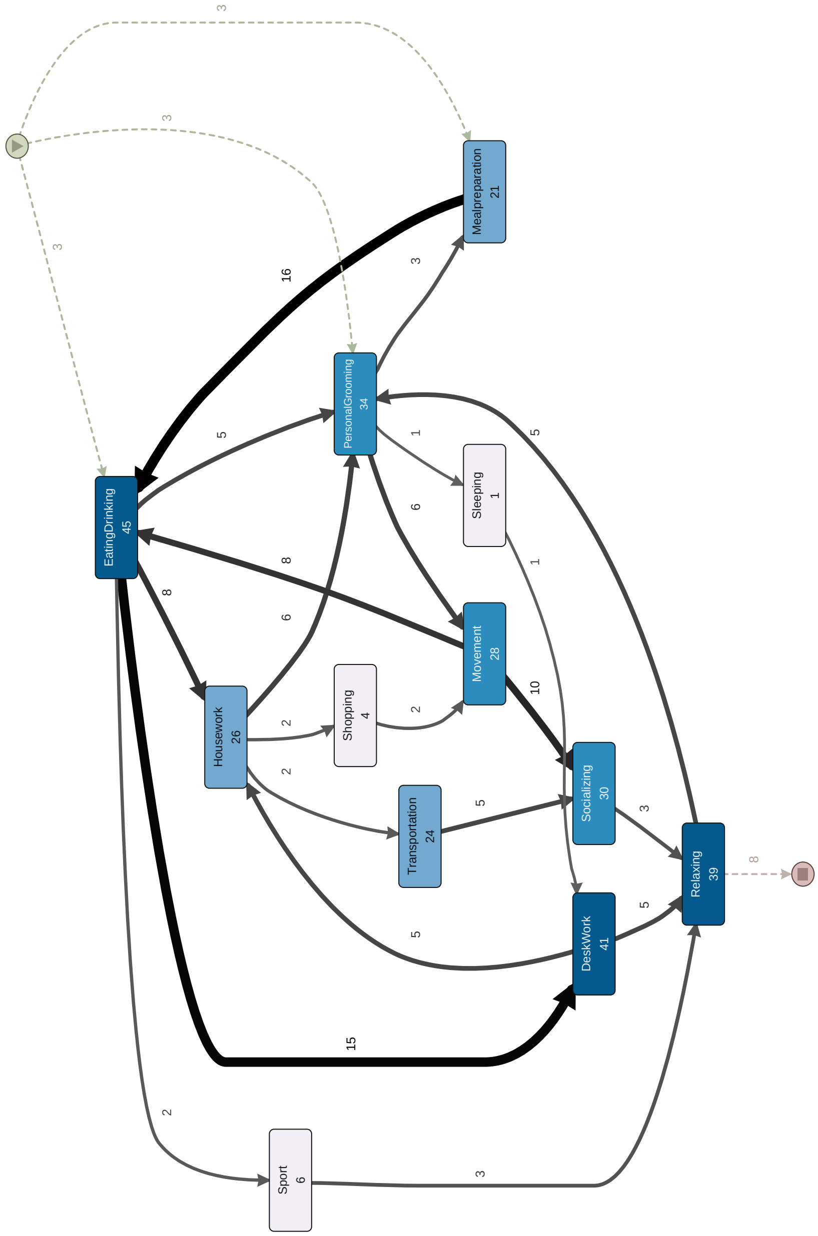
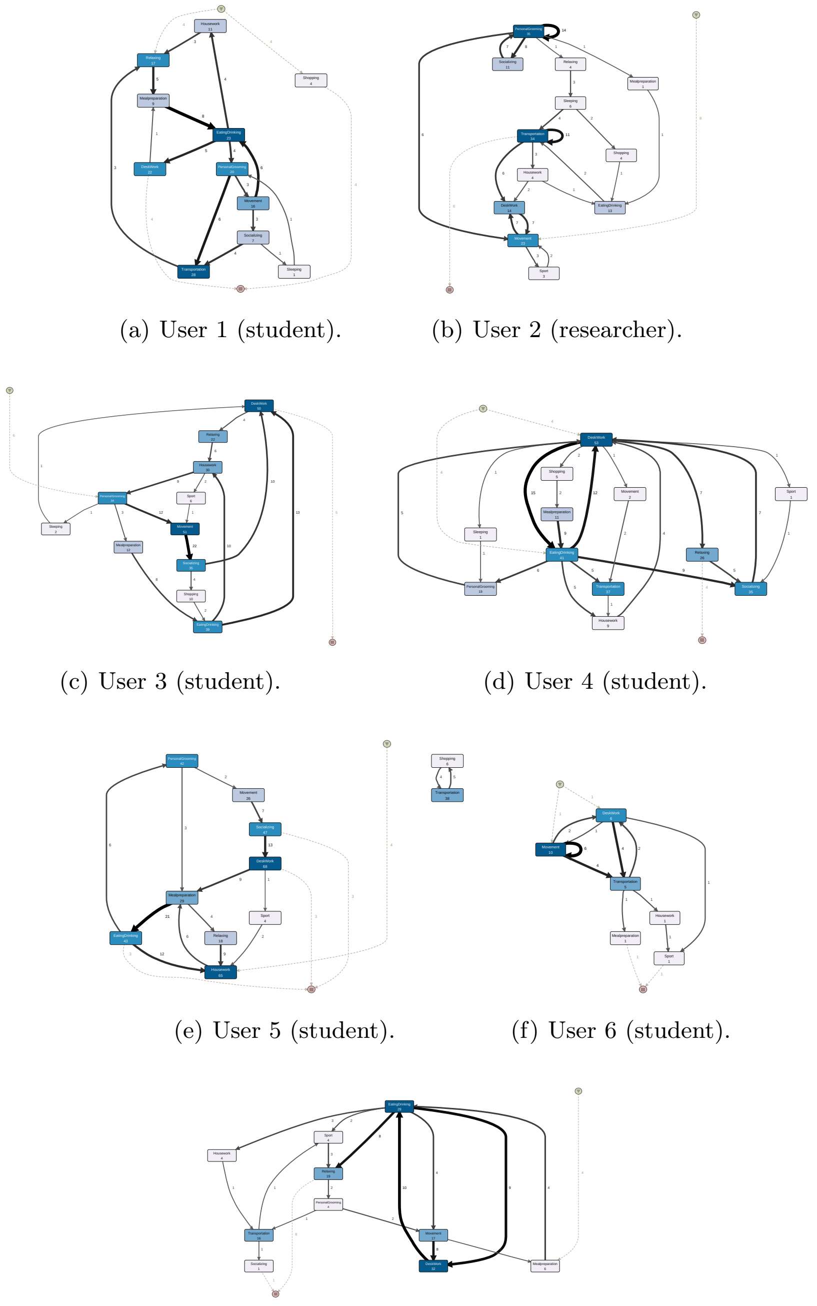


Fig. A.2. Main personal activity for all the users during the weekend days (17 cases).



(a) User 1 (student).

(b) User 2 (researcher).

(c) User 3 (student).

(d) User 4 (student).

(e) User 5 (student).

(f) User 6 (student).

(g) User 7 (worker).

Fig. A.3. Main personal activity by users.

References

1. American Heart Association. http://www.heart.org. Last Access: 29.04.2015.
2. Steven N Blair and Tim S Church. The fitness, obesity, and health equation: is physical activity the common denominator? *Jama*, 292(10):1232–1234, 2004.
3. RP J. C. Bose and Wil MP van der Aalst. Process diagnostics using trace alignment: Opportunities, issues, and challenges. *Inf. Syst.*, 37(2):117–141, 2012.
4. Disco by Fluxicon. https://fluxicon.com/disco/. Last Access: 29.04.2015.
5. M. De Leoni and W.M.P. van der Aalst. Aligning event logs and process models for multi-perspective conformance checking: An approach based on integer linear programming. In *Business Process Management, China*, pages 113–129, 2013.
6. F. Friedrich, J. Mendling, and F. Puhlmann. Process model generation from natural language text. In *Advanced Information Systems Engineering - 23rd International Conference, CAiSE 2011, London, UK, 2011. Proceedings*, pages 482–496, 2011.
7. Fosca Giannotti, Mirco Nanni, Fabio Pinelli, and Dino Pedreschi. Trajectory pattern mining. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 330–339. ACM, 2007.
8. C.W. Günther and W.M.P. van der Aalst. Fuzzy mining - adaptive process simplification based on multi-perspective metrics. In *BPM*, pages 328–343, 2007.
9. O.D. Lara and M.A. Labrador. A survey on human activity recognition using wearable sensors. *Communications Surveys & Tutorials, IEEE*, 15(3):1192–1209, 2013.
10. Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Discovering block-structured process models from event logs containing infrequent behaviour. In *Business Process Management Workshops - BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers*, pages 66–78, 2013.
11. Zhenhui Li. Spatiotemporal pattern mining: Algorithms and applications. In *Frequent Pattern Mining*, pages 283–306. Springer, 2014.
12. Nishkam Ravi, Nikhil Dandekar, Preetham Mysore, and Michael L Littman. Activity recognition from accelerometer data. In *AAAI*, volume 5, pages 1541–1546, 2005.
13. Chaoming Song, Zehui Qu, Nicholas Blumm, and Albert-László Barabási. Limits of predictability in human mobility. *Science*, 327(5968):1018–1021, 2010.
14. Hsiao-Ping Tsai, De-Nian Yang, and Ming-Syan Chen. Mining group movement patterns for tracking moving objects efficiently. *Knowledge and Data Engineering, IEEE Transactions on*, 23(2):266–281, 2011.
15. Wil M. P. van der Aalst. *Process Mining - Discovery, Conformance and Enhancement of Business Processes.* Springer, 2011.
16. Wil M. P. van der Aalst. Process cubes: Slicing, dicing, rolling up and drilling down event data for process mining. In *Asia Pacific Business Process Management, Beijing, China*, pages 1–22, 2013.
17. Wil M. P. van der Aalst, H. T. de Beer, and Boudewijn F. van Dongen. Process mining and verification of properties: An approach based on temporal logic. In *CoopIS, Cyprus*, pages 130–147, 2005.
18. A.J.M.M. Weijters, W.M.P. van der Aalst, and A.K. Alves de Medeiros. Process mining with the heuristics miner-algorithm. Technical Report WP 166, BETA Working Paper Series, Eindhoven University of Technology, 2006.
19. A.Y. Yang, S. Iyengar, S. Sastry, R. Bajcsy, P. Kuryloski, and R. Jafari. Distributed segmentation and classification of human actions using a wearable motion sensor network. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on*, pages 1–8. IEEE, 2008.
20. Yu Zheng. Trajectory data mining: an overview. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 6(3):29, 2015.

ILP-Based Process Discovery Using Hybrid Regions

S.J. van Zelst, B.F. van Dongen, and W.M.P. van der Aalst

Department of Mathematics and Computer Science
Eindhoven University of Technology, The Netherlands
{s.j.v.zelst,b.f.v.dongen,w.m.p.v.d.aalst}@tue.nl

Abstract. The *language-based theory of regions*, stemming from the area of *Petri net synthesis*, forms a fundamental basis for Integer Linear Programming (ILP)-based process discovery. Based on example behavior in an event log, a process model is derived that aims to describe the observed behavior. Building on top of the existing ILP-formulation, we present a new ILP-based process discovery formulation that unifies two existing types of language-based regions and, additionally, we present a generalized ILP objective function that captures both region-types and helps us to find suitable process discovery results.

Keywords: Process mining, process discovery, integer linear programming, region theory

1 Introduction

Process Mining [1] forms the connecting element between business process modeling and data analysis. Its main aim is to extract knowledge from business process execution data originating from a variety of data sources, e.g., enterprise information systems, social media, embedded systems etc. Within process mining, three main branches are distinguished being *process discovery*, *conformance checking* and *process enhancement*. Within process discovery the aim is to, given an event log, *reconstruct* a process model. Within conformance checking the aim is to assess, given a process model and an event log, the *conformance* of the event log to the process model. Within process enhancement the aim is to further improve and/or align existing process models by combining the two aforementioned disciplines, e.g., identification and removal of bottlenecks within business processes.

The area of *Petri net synthesis* [2] is concerned with deciding whether there exists a Petri net that *exactly* describes a given *sequential behavioral system description*. A subclass of synthesis techniques is the area of *region theory* which is both defined on transition systems, known as *state-based region theory*, and on languages, known as *language-based region theory*.

Language-based region theory forms a fundamental basis for ILP-based process discovery [3]. Within process discovery however, a process model is typically valued w.r.t. four essential process mining quality dimensions being *replay-fitness*, *precision*, *generalization* and *simplicity* [4]. Applying traditional Petri net synthesis techniques in a process discovery context will result in models that have perfect replay-fitness, maximal precision, low generalization and often low simplicity.

In this paper we propose a unified approach w.r.t. ILP-based process discovery, based on the newly introduced concept of *hybrid variable-based regions*. Hybrid variable-based regions unite two existing language-based region definitions and allow us to vary the number of variables used within the ILP problems being solved. Therefore, we assess the impact of using a different number of variables on the average computation time of solving ILPs during ILP-based process discovery. Additionally we present a new generalized ILP objective function that supports hybrid variable-based regions and show that the objective function yields correct process discovery results.

The outline of this paper is as follows. In Section 2 we present basic preliminaries. In Section 3 we present language-based regions. In Section 4 we show how to adopt language-based regions within process discovery using ILP. In Section 5 we present a brief evaluation of the performance of the approach. Section 6 covers related work. Section 7 concludes the paper.

2 Preliminaries

2.1 Bags, Sequences and Vectors

Let $\mathbb{Z}$ denote the set of integers, let $\mathbb{N}$ denote the set of positive integers *including* 0 and let $\mathbb{N}^+$ denote the set of positive integers *excluding* 0. Let $\mathbb{R}$ denote the set of real numbers and let $\mathbb{R}^+$ denote the set of positive real numbers excluding 0.

Let S denote a set. Let us define a *bag* B as a function $B : S \to \mathbb{N}$. Notation-wise we write a bag as $[e_1^{n_1}, e_2^{n_2}, ..., e_3^{n_3}]$, where $e_i \in S$, $n_i \in \mathbb{N}^+$ and $e_i^{n_i} \equiv B(e_i) = n_i$. If for some element e, $B(e) = 1$, we omit its superscript. An empty bag is denoted as $\emptyset$. As an example let $S_1 = \{a, b, c, d\}$ and let B_1 denote a bag consisting of 3 a's, 5 b's, 1 c and 0 d's, i.e. $B_1 = [a^3, b^5, c]$. Element inclusion applies to bags as well, i.e. given $e \in S$ and $B(e) > 0$ then also $e \in B$. Thus we have $a \in B_1$ whereas $d \notin B_1$.

A *sequence* σ is a function relating positions to elements $e \in S$, i.e. $\sigma : \{1, 2, ..., k\} \to S$. An empty sequence is denoted as ϵ. We write every non-empty sequence as $\langle e_1, e_2, ..., e_k\rangle$. The set of all possible sequences over a set S is denoted as S^*. We define *concatenation* of sequences σ_1 and σ_2 as $\sigma_1 \cdot \sigma_2$, e.g., $\langle a, b\rangle \cdot \langle c, d\rangle = \langle a, b, c, d\rangle$. If we are given a set X of sequences over S, i.e., $X \subseteq S^*$, we define X's *prefix-closure* as $\overline{X} = \{\sigma_1 \in S^* | \exists_{\sigma_2 \in S^*}(\sigma_1 \cdot \sigma_2 \in X)\}$. Let $X \subseteq S^*$, X is *prefix-closed* if $X = \overline{X}$. Let $X \subseteq S^*$ and let $B_X : X \to \mathbb{N}$ be a bag, we define B_X's prefix closure as $\overline{B}_X : \overline{X} \to \mathbb{N}$

$$\overline{B}_X(\sigma) = B(\sigma) + \sum_{\sigma \cdot \langle e\rangle \in \overline{X}} \overline{B}_X(\sigma \cdot \langle e\rangle)$$

Thus, for set $X_1 = \{\langle a, b\rangle, \langle a, c\rangle\}$ we have $\overline{X}_1 = \{\epsilon, \langle a\rangle, \langle a, b\rangle, \langle a, c\rangle\}$ whereas for bag $B_2 = [\langle a, b\rangle^5, \langle a, c\rangle^3]$ we have $\overline{B}_2 = [\epsilon^8, \langle a\rangle^8, \langle a, b\rangle^5, \langle a, c\rangle^3]$.

Let S be a set on which we can pose some total order and let $R \subseteq \mathbb{R}$ be a range of values. Vectors are denoted as $\vec{z} \in R^{|S|}$, where $\vec{z}(e) \in R$ and $e \in S$. A vector $\vec{z}$ represents a column vector. For vector multiplication we assume vectors to agree on their indices. Thus, given a totally ordered set $S = \{e_1, e_2, ..., e_n\}$ and $\vec{z}_1, \vec{z}_2 \in R^{|S|}$ we have $\vec{z}_1^{\intercal}\vec{z}_2 = \sum_{i \in \{1,2,...,n\}} \vec{z}_1(e_i)\vec{z}_2(e_i)$. *Parikh vectors* represent the number of occurrences of a certain element within a sequence. A Parikh vector is a function

$\vec{p} : S^* \rightarrow \mathbb{N}^{|S|}$ with $\vec{p}(\sigma) = (\sigma_{\upharpoonright e_1}, \sigma_{\upharpoonright e_2}, ..., \sigma_{\upharpoonright e_n})$ where $\sigma_{\upharpoonright e_i} = |\{j \mid 1 \leq j \leq |\sigma|, \sigma(j) = e_i\}|$. As an example consider $S = \{a, b, c, d\}$, $\sigma_1 = \langle a, b, d \rangle$ and $\sigma_2 = \langle a, c, d \rangle$. We have $\vec{p}(\sigma_1)^\intercal = (1, 1, 0, 1)$ and $\vec{p}(\sigma_2)^\intercal = (1, 0, 1, 1)$. Note that $\sigma_{1\upharpoonright b} = 1$ whereas $\sigma_{2\upharpoonright b} = 0$. Given a Parikh vector $\vec{p} : S^* \rightarrow \mathbb{N}^{|S|}$ and a set $Q \subseteq S$, we define $\vec{p}_Q : S^* \rightarrow \mathbb{N}^{|Q|}$. Thus given $S = \{a, b, c, d\}$ and $\sigma_1 = \langle a, b, d \rangle$ we have $\vec{p}_{\{a,c,d\}}(\sigma_1)^\intercal = (1, 0, 1)$.

2.2 Event Logs and Petri Nets

The main goal of process discovery is to describe the observed behavior within an event log by means of a process model. Thus, event logs act as the input of process discovery whereas some process model acts as the output of process discovery. An abstract example of an event log is presented in Table 1. Often more data attributes are available in an event log, e.g., *customer id*, *credit balance* etc. In this paper we focus on the sequential ordering of *activities* w.r.t. *cases*, i.e. the *control-flow perspective*.

Definition 1 (Event log). *Let A be a set of activities, an event log L is a bag of sequences over A, i.e., $L : A^* \rightarrow \mathbb{N}$.*[1]

A sequence of activities $\sigma \in L$ is referred to as a *trace*. We assume that any given set of activities A is totally ordered (or we are able to trivially pose a total ordering on A).

We consider Petri nets to describe process models. A Petri net is a bipartite graph consisting of a set of vertices called *places* and a set of vertices called *transitions*. Arcs (directed edges) are connecting places and transitions and have an associated positive weight.

Definition 2 (Petri net). *A Petri net is a 3-tuple $N = (P, T, W)$, where P is a set of places and T is a set of transitions with $P \cap T = \emptyset$. W is the weighted flow relation of N, $W : (P \times T) \cup (T \times P) \rightarrow \mathbb{N}$.*

For a given node $x \in P \cup T$, the pre-set of x in N is defined as $\bullet x = \{y \mid W(y, x) > 0\}$. Correspondingly $x \bullet = \{y \mid W(x, y) > 0\}$ denotes the post-set of x in N. Graphically we represent places as *circles* and transitions as *boxes*. For every $(x, y) \in (P \times T) \cup (T \times P)$ with $W(x, y) > 0$ we draw an *arc* from x to y. If $W(x, y) > 1$

Table 1: An abstract example of an *event log*.

Case id	Activity id	Resource id	Time-stamp
c1	*a*	*Lucy*	*2015-01-05T09:13:37+00:00*
c2	*a*	*John*	*2015-01-05T13:37:25+00:00*
c2	*b*	*Pete*	*2015-01-06T13:14:15+00:00*
c2	*d*	*Lucy*	*2015-01-06T15:27:18+00:00*
c1	*c*	*Pete*	*2015-01-07T14:28:56+00:00*
c1	*d*	*John*	*2015-01-07T15:30:00+00:00*
⋮	⋮	⋮	⋮

[1] In practice we extract an event log L from some information system. Consequently A is implicitly defined by the event log, i.e., A only consists of events that are actually present L.

we denote the arc's weight $W(x, y)$ on top of the arc from x to y. A Petri net is *pure* if it does not contain self-loops, i.e., if $W(x, y) > 0$ then $W(y, x) = 0$. An example (pure) Petri net is depicted in Figure 1.

Definition 3 (Marked Petri net). *Let $N = (P, T, W)$ be a Petri net. A marking of N is a bag over N's places, i.e. $M : P \rightarrow \mathbb{N}$. A marked Petri net is a pair (N, M_0), where M_0 represents N's initial marking.*

Let $N = (P, T, W)$ be a Petri net and let M be a marking of N. A transition $t \in T$ is *enabled*, denoted $(N, M)[t\rangle$, if and only if $\forall_{p \in \bullet t}(M(p) \geq W(p, t))$. Graphically we represent a marking by drawing exactly a place's marking-multiplicity number of dots inside the place (e.g. p_1 in Figure 1 with $M_0 = [p_1]$). If a transition t is enabled in (N, M), t may *fire*, resulting in a new marking M'. When t fires, denoted as $(N, M)[t\rangle(N, M')$, we have $\forall_{p \in P}(M'(p) = M(p) - W(p, t) + W(t, p))$. For example in Figure 1 we have $(N_1, [p_1])[a\rangle(N_1, [p_2])$.

Definition 4 (Firing sequences). *Let $N = (P, T, W)$ be a Petri net and let (N, M_0) be a corresponding marked Petri net. Sequence $\sigma = \langle t_1, t_2, ..., t_n\rangle \in T^*$ is a firing sequence of (N, M_0), written as $(N, M_0)[\sigma\rangle(N, M_n)$ if and only if for $n = |\sigma|$ there exist markings $M_0, M_1, M_2, ..., M_n$ such that $(N, M_0)[t_1\rangle(N, M_1)$, $(N, M_1)[t_2\rangle(N, M_2), ..., (N, M_{n-1})[t_n\rangle(N, M_n)$.*

Note that infinitely many firing sequences can exist given a Petri net. Some example firing sequences of the Petri net depicted in Figure 1 are: $(N_1, [p_1])[\langle a\rangle\rangle(N_1, [p_2])$, $(N_1, [p_1])[\langle a, b\rangle\rangle(N_1, [p_3])$ and $(N_1, [p_1])[\langle a, c, d, e, f, e, f, e, g\rangle\rangle(N_1, \emptyset)$. The set of all possible firing sequences in a Petri net N is called $N's$ language, i.e., all sequences $\sigma \in T^*$ s.t. $(N, M_0)[\sigma\rangle(N, M_i)$. N's language is denoted as $\mathcal{L}(N) \subseteq T^*$ and is prefix-closed.

Consider Figure 1 and event log $L_1 = [\langle a, b, d, e, g\rangle^{10}, \langle a, c, d, e, g\rangle^{9}, \langle a, b, d, e, f, e, g\rangle^{11}, \langle a, c, d, e, f, e, g\rangle^{8}]$ over $A_1 = \{a, b, c, d, e, f, g\}$. Clearly we have $\overline{L_1} \subset \mathcal{L}(N_1)$, and thus replay-fitness of L_1 on N_1 is perfect. We will use N_1, A_1 and L_1 throughout the paper as a running-example.

3 Regions

The concept of regions forms the basis of region theory within Petri net synthesis. Given an event log L over a set of activities A, language-based regions are used to represent

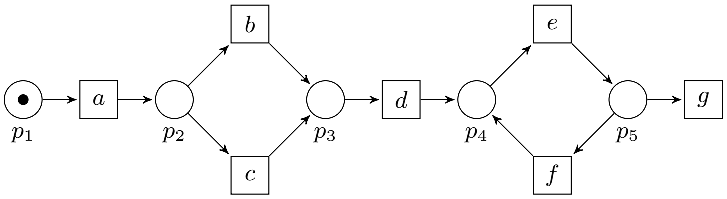


Fig. 1: A pure Petri net $N_1 = (P_1, T_1, W_1)$ with $P_1 = \{p_1, p_2, ..., p_5\}, T_1 = \{a, b, ..., g\}$ and $W_1(p_1, a) = W_1(a, p_2) = ... = W_1(p_5, g) = 1$.

places in a resulting Petri net $N = (P, A, W)$. A language-based region maps every $a \in A$ to an integer representing arc-weight. For such mapping over A to be a region we pose the restriction that the corresponding place $p \in P$ should not block any $\sigma \in \overline{L}$, i.e., $\sigma \in \overline{L} \Rightarrow \sigma \in \mathcal{L}(N)$. We identify two basic definitions of language-based regions which we classify as *single variable-based regions* and *dual variable-based regions*. The main difference is the number of variables used to define a region.

Definition 5 (Single variable-based regions). *Let L be an event log over a set of activities A. Let $m \in \mathbb{N}$ and let $\vec{v} \in \mathbb{Z}^{|A|}$. A pair $r = (m, \vec{v})$ is a single variable-based region iff:*

$$\forall_{\sigma \in \overline{L}}(m + \vec{p}(\sigma)^{\intercal}\vec{v} \geq 0) \tag{3.1}$$

Let $R^s(\overline{L})$ denote the set of all possible single variable-based regions of $\overline{L}$. Equation 3.1 generates a set of linear inequalities, i.e. applying Equation 3.1 on L_1 yields:

$$\begin{array}{ccc}
m & \geq 0 & \epsilon \\
m + \vec{v}(a) & \geq 0 & \langle a \rangle \\
m + \vec{v}(a) + \vec{v}(b) & \geq 0 & \langle a, b \rangle \\
\vdots & \vdots & \vdots \\
m + \vec{v}(a) + \vec{v}(b) + \vec{v}(d) + 2\vec{v}(e) + \vec{v}(f) + \vec{v}(g) & \geq 0 & \langle a, b, d, e, f, e, g \rangle \\
m + \vec{v}(a) + \vec{v}(c) + \vec{v}(d) + 2\vec{v}(e) + \vec{v}(f) + \vec{v}(g) & \geq 0 & \langle a, c, d, e, f, e, g \rangle
\end{array}$$

Single variable-based regions use one *single* decision variable for each $a \in A$, represented by $\vec{v} \in \mathbb{Z}^{|A|}$. Expressing a single variable-based region $r = (m, \vec{v})$ as a place $p \in P$ in a marked net (N, M_0) with $N = (P, A, W)$ is straightforward. We have $M_0(p) = m$, if $\vec{v}(a) > 0$ then $W(a, p) = \vec{v}(a)$, if $\vec{v}(a) < 0$ then $W(p, a) = -\vec{v}(a)$ and if $\vec{v}(a) = 0$ then $W(a, p) = W(p, a) = 0$. Consider place p_2 depicted in Figure 1 which can be represented as a single variable-based region $r = (m, (\vec{v}(a), \vec{v}(b), ..., \vec{v}(g))) = (0, (1, -1, -1, 0, 0, 0, 0))$. Note that for each inequality generated by Equation 3.1, place p_2 has a value of at least 0 and hence is a member of $R^s(\overline{L})$.

Single variable-based regions only allow us to synthesize/discover pure Petri nets. As a consequence we can not find self-loops, i.e. we can not find a place p in a resulting net $N = (P, A, W)$ s.t. $p \in \bullet a \cap a \bullet$, for any $a \in A$. A lot of workflow patterns [5] in fact exhibit places that include self-loops, e.g. milestone patterns, mutual-exclusion patterns, priority patterns etc. Hence, we define *dual variable-based regions* which explicitly allow us to distinguish between incoming and outgoing arcs.

Definition 6 (Dual variable-based regions). *Let L be an event log over a set of activities A. Let $m \in \mathbb{N}$ and $\vec{x}, \vec{y} \in \mathbb{N}^{|A|}$. A triple $r = (m, \vec{x}, \vec{y})$ is a dual variable-based region iff:*

$$\forall_{\sigma = \sigma' \cdot \langle a \rangle \in \overline{L}}(m + \vec{p}(\sigma')^{\intercal}\vec{x} - \vec{p}(\sigma)^{\intercal}\vec{y} \geq 0) \tag{3.2}$$

Let $R^d(\overline{L})$ denote the set of all possible dual variable-based regions of $\overline{L}$. Like Definition 5, Definition 6 generates a set of linear inequalities. Applying Definition 6 on L_1 yields:

$$\begin{array}{ccc}
m - \vec{y}(a) & \geq 0 & \langle a \rangle \\
m - \vec{y}(a) + \vec{x}(a) - \vec{y}(b) & \geq 0 & \langle a, b \rangle \\
m - \vec{y}(a) + \vec{x}(a) - \vec{y}(c) & \geq 0 & \langle a, c \rangle \\
\vdots & \vdots & \vdots \\
m - \vec{y}(a) + \vec{x}(a) - \vec{y}(b) + \vec{x}(b) - \vec{y}(d) + \vec{x}(d) - 2\vec{y}(e) + 2\vec{x}(e) - \vec{y}(f) + \vec{x}(f) - \vec{y}(g) & \geq 0 & \langle a, b, d, e, f, e, g \rangle \\
m - \vec{y}(a) + \vec{x}(a) - \vec{y}(c) + \vec{x}(c) - \vec{y}(d) + \vec{x}(d) - 2\vec{y}(e) + 2\vec{x}(e) - \vec{y}(f) + \vec{x}(f) - \vec{y}(g) & \geq 0 & \langle a, c, d, e, f, e, g \rangle
\end{array}$$

Dual variable-based regions use two decision variables per $a \in A$, represented by $\vec{x}, \vec{y} \in \mathbb{N}^{|A|}$. The variables allow us to distinguish between *incoming arcs* and *outgoing arcs* when translating regions to Petri nets. Expressing a dual variable-based region $r = (m, \vec{x}, \vec{y})$ as a place $p \in P$ in marked net (N, M_0) with $N = (P, A, W)$ is again straightforward. We have $M_0(p) = m$, $W(a, p) = \vec{x}(a)$ and $W(p, a) = \vec{y}(a)$. Again consider place p_2 depicted in Figure 1 which can be represented as a dual variable-based region $r = (m, (\vec{x}(a), \vec{x}(b), ..., \vec{x}(g)), (\vec{y}(a), \vec{y}(b), ..., \vec{y}(g))) = (0, (1, 0, 0, 0, 0, 0, 0), (0, 1, 1, 0, 0, 0, 0))$. Verify that for each linear in-equality generated by Definition 6, place p_2 has a value of at least 0 and hence is a member of $R^d(\overline{L})$.

4 Hybrid Variable-Based Regions in Process Discovery

Using dual variable-based regions allows us to express non-pure places, i.e, self-loops, milestones etc. However, this type of regions uses roughly twice as many variables compared with single variable-based regions. To balance the number of variables used, though still enhance the possibility of finding non-pure Petri net structures, we introduce the new concept of hybrid regions, capturing both single and dual variable-based regions.

Definition 7 (Hybrid variable-based regions). *Let L be an event log over a set of activities A. Let $A_s, A_d \subseteq A$ be two sets of activities with $A_s \cup A_d = A$ and $A_s \cap A_d = \emptyset$. Let $m \in \mathbb{N}$, $\vec{v} \in \mathbb{Z}^{|A_s|}$ and $\vec{x}, \vec{y} \in \mathbb{N}^{|A_d|}$. A quadruple $r = (m, \vec{v}, \vec{x}, \vec{y})$ is a hybrid variable-based region iff:*

$$\forall_{\sigma = \sigma' \cdot \langle a \rangle \in \overline{L}} (m + \vec{p}_{A_s}(\sigma)^\intercal \vec{v} + \vec{p}_{A_d}(\sigma')^\intercal \vec{x} - \vec{p}_{A_d}(\sigma)^\intercal \vec{y} \geq 0) \tag{4.1}$$

Given a set of activities A and two sets of activities $A_s, A_d \subseteq A$ with $A_s \cup A_d = A$ and $A_s \cap A_d = \emptyset$, we refer to a *hybrid partition* of A. If we choose $A_d = A$, Equation 4.1 is equal to Equation 3.2. If we choose $A_s = A$, Equation 4.1 can be reformulated as:

$$\forall_{\sigma \in \overline{L} \setminus \{\epsilon\}} (m + \vec{p}(\sigma)^\intercal \vec{v} \geq 0) \tag{4.2}$$

Equation 4.2 does not equal Equation 3.1, however, as $\vec{p}(\epsilon) = \vec{0}$ and $m \in \mathbb{N}$, the equations are equivalent in this context.

Note that the set of hybrid variable-based regions, i.e. the set of variable assignments that adhere to Definition 7 depends on the hybrid partition of A into A_s and A_d. Therefore, we let A_s act as a parameter for the set of feasible hybrid variable-based regions. Let $R^h_{A_s}(\overline{L})$ denote the set of all possible hybrid variable-based regions of $\overline{L}$ where A_s represent a hybrid partition of A. Note $R^h_A(\overline{L}) = R^s(\overline{L})$ and $R^h_\emptyset(\overline{L}) = R^d(\overline{L})$. Region $r = (0, \vec{0}, \vec{0}, \vec{0})$ is deemed the *trivial region*.

All three language-based region definitions, i.e. single, dual and hybrid variable-based regions, provide means to accept and/or reject potential places in a to be constructed Petri net. In classical Petri net synthesis approaches, one keeps looking for feasible places until either $\overline{L} = \mathcal{L}(N)$, or if this is impossible, $\mathcal{L}(N) \setminus \overline{L}$ is minimized. Unfortunately, most models returned by classical Petri net synthesis techniques result in models that are unusable from a process discovery perspective. Hence, we need to relax the strict formal requirements posed on the relation between $\overline{L}$ and $\mathcal{L}(N)$.

Definition 8 (Hybrid variable-based process discovery ILP-formulation). *Let L be an event log over a set of activities A and let $A_s, A_d \subseteq A$ be a hybrid partition. Let $\mathbf{M}_s, \mathbf{M}_d, \mathbf{M}'_d$ be three matrices where $\mathbf{M}_s$ is an $|\overline{L}| \setminus \{\epsilon\} \times A_s$ matrix with $\mathbf{M}_s(\sigma, a) = \vec{p}(\sigma)(a)$ and $\mathbf{M}_d$, $\mathbf{M}'_d$ are two $|\overline{L}| \setminus \{\epsilon\} \times A_d$ matrices with $\mathbf{M}_d(\sigma, a) = \vec{p}(\sigma)(a)$ and $\mathbf{M}'_d(\sigma, a) = \vec{p}(\sigma')(a)$ (where $\sigma = \sigma' \cdot \langle a' \rangle \in \overline{L}$). Let $c_m \in \mathbb{R}$, $\vec{c_v} \in \mathbb{R}^{|A_s|}$ and $\vec{c_x}, \vec{c_y} \in \mathbb{R}^{|A_d|}$. The hybrid variable-based process discovery ILP-formulation, denoted $ILP^h_{\overline{L}}$, is defined as:*

$$
\begin{array}{rrl}
\textbf{\textit{minimize}} & z = c_m m + \vec{c_v}^\intercal \vec{v} + \vec{c_x}^\intercal \vec{x} + \vec{c_y}^\intercal \vec{y} & \textit{objective function} \\
\textbf{\textit{such that}} & m\vec{1} + \mathbf{M}_s \vec{v} + \mathbf{M}'_d \vec{x} - \mathbf{M}_d \vec{y} \geq \vec{0} & \textit{hybrid variable-based region} \\
\textbf{\textit{and}} & -\vec{1} \leq \vec{v} \leq \vec{1} & \textit{i.e. } \vec{v} \in \{-1, 0, 1\}^{|A_s|} \\
 & \vec{0} \leq \vec{x} \leq \vec{1} & \textit{i.e. } \vec{x} \in \{0, 1\}^{|A_d|} \\
 & \vec{0} \leq \vec{y} \leq \vec{1} & \textit{i.e. } \vec{y} \in \{0, 1\}^{|A_d|} \\
 & 0 \leq m \leq 1 & \textit{i.e. } m \in \{0, 1\}
\end{array}
$$

The ILP-fromulation presented in Definition 8 uses the set of linear in-equalities generated by Equation 4.1 within its constraint body. The formulation however binds $\vec{v}$ to $\{-1, 0, 1\}^{|A_s|}$ and $\vec{x}, \vec{y}$ to $\{0, 1\}^{|A_d|}$, i.e. the formulation only allows for discovering Petri nets with arc weights restricted to $\{0, 1\}$. Additionally it defines an objective function, i.e. $z = c_m m + \vec{c_v}^\intercal \vec{v} + \vec{c_x}^\intercal \vec{x} + \vec{c_y}^\intercal \vec{y}$, that maps each region to a real value, i.e. $z : R^h_{A_s}(\overline{L}) \rightarrow \mathbb{R}$. In general we are free to choose whatever objective function we like, however, using different objective functions will lead to different process discovery results. Choosing $c_m = 0$ and $\vec{c}_v = \vec{c}_x = \vec{c}_y = \vec{1}$ leads to arc minimization whereas maximizing the same objective leads to arc maximization, resulting in different places/regions found by the underlying ILP solver. The objective function proposed in [3], being a minimization function, tries to minimize the number of incoming arcs and maximize the number of outgoing arcs. The objective function can be defined in terms of $c_m, \vec{c_v}, \vec{c_x}$ and $\vec{c_y}$ as $c_m = |\overline{L}|$, $\vec{c_v} = \sum_{\sigma \in \overline{L}} \vec{p}_{A_s}(\sigma)$, $\vec{c_x} = \sum_{\sigma \in \overline{L}} \vec{p}_{A_d}(\sigma)$ and $\vec{c_y} = -\vec{c_x}$, i.e.:

$$
z(r) = \sum_{\sigma \in \overline{L}} (m + \vec{p}_{A_s}(\sigma)^\intercal \vec{v} + \vec{p}_{A_d}(\sigma)^\intercal (\vec{x} - \vec{y})) \tag{4.3}
$$

4.1 Optimizing Token Throughput

The objective function used within [3], i.e. Equation 4.3 is less generic as the one proposed in Definition 8. As shown in [3], it favors *minimal* regions. A minimal region is a region that is not expressible as the sum of two other regions. The objective function is solely based on the set-representation of the prefix-closure of an event log. However, an event log is a bag of traces and thus consists of information on trace frequency. We propose a generalized *prefix-closure-based* objective function that incorporates a word-based scaling function β. The scaling function β is required to map all sequences in the prefix-closure of the event log to some positive real value. The actual implementation is up to the user, although we present an instantiation of β that works well for process discovery. We show that the proposed generalized weighted prefix-closure-based hybrid region objective function favors minimal regions, given any scaling function β.

Definition 9 (Generalized weighted prefix-closure-based hybrid region objective function). *Let L be an event log over a set of activities A and let $A_s, A_d \subseteq A$ be a hybrid partition. Let $r = (m, \vec{v}, \vec{x}, \vec{y}) \in R^h_{A_s}(\overline{L})$ be a hybrid variable-based region and let β be a scaling function over $\overline{L}$, i.e. $\beta : \overline{L} \to \mathbb{R}^+$. The generalized weighted prefix-closure-based hybrid region objective function is instantiated as $c_m = \sum_{\sigma \in \overline{L}} \beta(\sigma)$, $\vec{c_v} = \sum_{\sigma \in \overline{L}} \beta(\sigma)\vec{p}_{A_s}(\sigma)$, $\vec{c_x} = \sum_{\sigma \in \overline{L}} \beta(\sigma)\vec{p}_{A_d}(\sigma)$ and $\vec{c_y} = -\vec{c_x}$, i.e.:*

$$z_\beta(r) = \sum_{\sigma \in \overline{L}} \beta(\sigma)(m + \vec{p}_{A_s}(\sigma)^\intercal \vec{v} + \vec{p}_{A_d}(\sigma)^\intercal (\vec{x} - \vec{y})) \tag{4.4}$$

Note that if we choose $\beta(\sigma) = 1$ for all $\sigma \in \overline{L}$, denoted β_1, we instantiate the generalized objective function as the objective function proposed in [3]. We denote this objective function as z_1, i.e. Equation 4.3.

To relate the behavior in a given event log to the objective function defined in Definition 9 we instantiate the scaling function β making use of the frequencies of the traces present in the event log, i.e. we let $\beta(\sigma) = \overline{L}(\sigma)$ leading to:

$$z_{\overline{L}}(r) = \sum_{\sigma \in \overline{L}} \overline{L}(\sigma)(m + \vec{p}_{A_s}(\sigma)^\intercal \vec{v} + \vec{p}_{A_d}(\sigma)^\intercal (\vec{x} - \vec{y})) \tag{4.5}$$

To assess the difference between z_1 and the newly proposed objective function $z_{\overline{L}}$, consider the Petri net depicted in Figure 2. Assume we are given some event log $L = [\langle a, b, d\rangle^5, \langle a, c, d\rangle^3]$. Let r_1 denote the hybrid variable-based region corresponding to place p_1, let r_2 correspond to p_2 and let r_3 correspond to p_3. In this case we have $z_1(r_1) = 1$, $z_1(r_2) = 1$ and $z_1(r_3) = 2$. On the other hand we have $z_{\overline{L}}(r_1) = z_{\overline{L}}(r_2) = z_{\overline{L}}(r_3) = 5 + 3 = 8$. Thus, using z_{β_L} leads to more intuitive objective values compared to using z_1 as $z_{\overline{L}}$ evaluates to the absolute number of discrete time-steps a token would remain in the corresponding place when replaying log L w.r.t. the place.

In [3] it is shown that the objective function used favors minimal regions. The proof cannot directly be adapted to hold for hybrid variable-based regions. Moreover it does not provide means to show that any arbitrary instantiation of z_β favors minimal hybrid variable-based regions. Here we show that any instantiation of the generalized weighted prefix-closure-based hybrid region objective function with some scaling function $\beta : \overline{L} \to \mathbb{R}^+$ favors minimal hybrid variable-based regions. We first show that the objective value of a non-minimal hybrid region equals the sum of the two minimal regions defining it after which we show that the given objective function maps each region to some positive real value, i.e. $rng(z_\beta) \subseteq \mathbb{R}^+$.

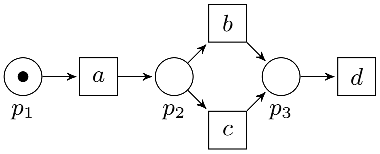


Fig. 2: A simple Petri net N with $\mathcal{L}(N) = \{\epsilon, \langle a\rangle, \langle a, b\rangle, \langle a, c\rangle\langle a, b, d\rangle, \langle a, c, d\rangle\}$.

Lemma 1 (Objective value composition of non-minimal regions). *Let L be an event log over a set of activities A and let $A_s, A_d \subseteq A$ be a hybrid partition. Let $r_1 = (m_1, \vec{v_1}, \vec{x}_1, \vec{y_1})$, $r_2 = (m_2, \vec{v_2}, \vec{x}_2, \vec{y_2})$ and $r_3 = (m_1{+}m_2, \vec{v}_1{+}\vec{v}_2, \vec{x}_1{+}\vec{x}_2, \vec{y}_1{+}\vec{y}_2)$ with r_1, $r_2, r_3 \in R^h_{A_s}(\overline{L})$. Let $z_\beta : R^h_{A_s}(\overline{L}) \rightarrow \mathbb{R}$ where z_β is an instantiation of the generalized weighted objective function as defined in Definition 9, then $z_\beta(r_3) = z_\beta(r_1) + z_\beta(r_2)$.*

Proof (By definition of z_β). Let us denote $z_\beta(r_3)$:

$$\sum_{\sigma \in \overline{L}} \beta(\sigma)((m_1 + m_2) + \vec{p}_{A_s}(\sigma)^\intercal(\vec{v_1} + \vec{v_2}) + \vec{p}_{A_d}(\sigma)^\intercal((\vec{x_1} + \vec{x_2}) - (\vec{y_1} + \vec{y_2})))$$

$$\sum_{\sigma \in \overline{L}} \beta(\sigma)(m_1{+}\vec{p}_{A_s}(\sigma)^\intercal\vec{v_1}{+}\vec{p}_{A_d}(\sigma)^\intercal(\vec{x_1}{-}\vec{y_1}){+}m_2{+}\vec{p}_{A_s}(\sigma)^\intercal\vec{v_2}{+}\vec{p}_{A_d}(\sigma)^\intercal(\vec{x_2}{-}\vec{y_2}))$$

$$\sum_{\sigma \in \overline{L}} \beta(\sigma)(m_1 + \vec{p}_{A_s}(\sigma)^\intercal\vec{v_1} + \vec{p}_{A_d}(\sigma)^\intercal(\vec{x_1} - \vec{y_1})) +$$
$$\sum_{\sigma \in \overline{L}} \beta(\sigma)(m_2 + \vec{p}_{A_s}(\sigma)^\intercal\vec{v_2} + \vec{p}_{A_d}(\sigma)^\intercal(\vec{x_2} - \vec{y_2}))$$

Clearly $z_\beta(r_3) = z_\beta(r_1) + z_\beta(r_2)$. □

Lemma 1 shows that the value of z_β for a non-minimal region equals the sum of the z_β values of the two regions it is composed of. If we additionally show that z_β can only evaluate to positive values, we show that z_β favors minimal hybrid variable-based regions.

Lemma 2 (Range of z_β is strictly positive). *Let L be an event log over a set of activities A and let $A_s, A_d \subseteq A$ be a hybrid partition. Let $r = (m, \vec{v}, \vec{x}, \vec{y})$ be a non-trivial region, i.e., $r \in R^h_{A_s}(\overline{L})$. If we let z_β be any instantiation of the generalized weighted objective function as defined in Definition 9, then $z_\beta : R^h_{A_s}(\overline{L}) \rightarrow \mathbb{R}^+$.*

Proof (By showing $z_\beta(r) > 0, \forall r \in R^h_{A_s}(\overline{L})$). Let $r = (m, \vec{v}, \vec{x}, \vec{y})$ be a non-trivial hybrid variable-based region, i.e. $r \in R^h_{A_s}(\overline{L})$. Because $r \in R^h_{A_s}(\overline{L})$ we have

$$\forall_{\sigma = \sigma' \cdot \langle a \rangle \in \overline{L} \setminus \{\epsilon\}}(m + \vec{p}_{A_s}(\sigma)^\intercal\vec{v} + \vec{p}_{A_d}(\sigma')^\intercal\vec{x} - \vec{p}_{A_d}(\sigma)^\intercal\vec{y} \geq 0) \tag{4.6}$$

Note that each Parikh value of an activity $a \in A$ given a sequence σ, i.e. $\vec{p}(\sigma)(a)$, is greater or equal than the Parikh value of a, given σ's prefix, i.e.,:

$$\forall_{\sigma = \sigma' \cdot \langle a' \rangle \in \overline{L}, a \in A}(\vec{p}(\sigma)(a) \geq \vec{p}(\sigma')(a)) \tag{4.7}$$

Using Equation 4.7 we can substitute $\vec{p}_{A_d}(\sigma')^\intercal\vec{x}$ with $\vec{p}_{A_d}(\sigma)^\intercal\vec{x}$ in Equation 4.6:

$$\forall_{\sigma = \sigma' \cdot \langle a \rangle \in \overline{L} \setminus \{\epsilon\}}(m + \vec{p}_{A_s}(\sigma)^\intercal\vec{v} + \vec{p}_{A_d}(\sigma)^\intercal(\vec{x} - \vec{y}) \geq 0) \tag{4.8}$$

Combining $rng(\beta) \subseteq \mathbb{R}^+$, $\vec{p}(\epsilon) = \vec{0}$ and $m \in \mathbb{N}$ with Equation 4.8 we find $z_\beta(r) \geq 0$:

$$\sum_{\sigma \in \overline{L}} \beta(\sigma)(m + \vec{p}_{A_s}(\sigma)^\intercal\vec{v} + \vec{p}_{A_d}(\sigma)^\intercal(\vec{x} - \vec{y})) \geq 0 \tag{4.9}$$

If $m > 0$ then $\beta(\epsilon)(m+\vec{p}_{A_s}(\epsilon)^\intercal\vec{v}+\vec{p}_{A_d}(\epsilon)^\intercal(\vec{x}-\vec{y})) > 0$. Combined with Equations 4.8 and 4.9 leads to $z_\beta(r) > 0$.

Observe that if $m = 0$ then for r to be a ***non-trivial*** *hybrid variable-based region, i.e. $r \in R^h_{A_s}(\overline{L})$, either (I). $\exists a \in A_s$ s.t. $\vec{v}(a) > 0$ or (II). $\exists a \in A_d$ s.t. $\vec{x}(a) > 0$.*

(I). Let $m = 0$ and $a \in A_s$ s.t. $\vec{v}(a) > 0$. We know $\exists\sigma = \sigma' \cdot \langle a\rangle \in \overline{L}$. Because $r \in R^h_{A_s}(\overline{L})$ (using Equation 4.8) we have:

$$m + \vec{p}_{A_s}(\sigma)^\intercal\vec{v} + \vec{p}_{A_d}(\sigma)^\intercal(\vec{x}-\vec{y}) \geq 0$$

If $\sigma' = \epsilon$, we have $\vec{p}_{A_d}(\sigma) = \vec{0}$ and $\vec{p}_{A_s}(\sigma)^\intercal\vec{v} = \vec{v}(a)$ hence we deduce:

$$m + \vec{p}_{A_s}(\sigma)^\intercal\vec{v} + \vec{p}_{A_d}(\sigma)^\intercal(\vec{x}-\vec{y}) > 0$$

Combining this with Equations 4.8 and 4.9 yields $z_\beta(r) > 0$.

If $\sigma' \neq \epsilon$ we have (using Equation 4.8):

$$m + \vec{p}_{A_s}(\sigma')^\intercal\vec{v} + \vec{p}_{A_d}(\sigma')^\intercal(\vec{x}-\vec{y}) \geq 0$$

Observe that $\vec{p}_{A_s}(\sigma)^\intercal\vec{v} = \vec{p}_{A_s}(\sigma')^\intercal\vec{v} + \vec{v}(a)$, together with $\vec{p}_{A_d}(\sigma') = \vec{p}_{A_d}(\sigma)$ leads us to reformulate this to:

$$m + \vec{p}_{A_s}(\sigma)^\intercal\vec{v} - \vec{v}(a) + \vec{p}_{A_d}(\sigma)^\intercal(\vec{x}-\vec{y}) \geq 0$$

$$m + \vec{p}_{A_s}(\sigma)^\intercal\vec{v} + \vec{p}_{A_d}(\sigma)^\intercal(\vec{x}-\vec{y}) \geq \vec{v}(a)$$

Combining this with Equations 4.8 and 4.9 yields $z_\beta(r) > 0$.

(II) Let $m = 0$ and $a \in A_d$ s.t. $\vec{x}(a) > 0$. We know $\exists\sigma = \sigma' \cdot \langle a\rangle \in \overline{L}$. Because $r \in R^h_{A_s}(\overline{L})$ (using Equation 4.6) we have:

$$m + \vec{p}_{A_s}(\sigma)^\intercal\vec{v} + \vec{p}_{A_d}(\sigma')^\intercal\vec{x} - \vec{p}_{A_d}(\sigma)^\intercal\vec{y} \geq 0$$

Observe that $\vec{p}_{A_d}(\sigma)^\intercal\vec{x} = \vec{p}_{A_d}(\sigma')^\intercal\vec{x} + \vec{x}(a)$ and thus:

$$m + \vec{p}_{A_s}(\sigma)^\intercal\vec{v} + \vec{p}_{A_d}(\sigma)^\intercal(\vec{x}-\vec{y}) \geq \vec{x}(a)$$

Combining this with Equations 4.8 and 4.9 yields $z_\beta(r) > 0$. □

By combining Lemma 1 and Lemma 2 we can easily show that any instantiation of z_β favors minimal regions.

Theorem 1 (Any instantiation of z_β favors minimal regions). *Let L be an event log over a set of activities A and let $A_s, A_d \subseteq A$ be a hybrid partition. Let $r_1 = (m_1, \vec{v}_1, \vec{x}_1, \vec{y}_1)$, $r_2 = (m_2, \vec{v}_2, \vec{x}_2, \vec{y}_2)$ and $r_3 = (m_1 + m_2, \vec{v}_1 + \vec{v}_2, \vec{x}_1 + \vec{x}_2, \vec{y}_1 + \vec{y}_2)$ be three non-trivial regions, i.e., $r_1, r_2, r_3 \in R^h_{A_s}(\overline{L})$. For any $z_\beta : R^h_{A_s}(\overline{L}) \to \mathbb{R}$ being an instantiation of the generalized weighted objective function as defined in Definition 9: $z_\beta(r_3) > z_\beta(r_1)$ and $z_\beta(r_3) > z_\beta(r_2)$.*

Proof (By composition of Lemma 1 and Lemma 2). By Lemma 1 we know $z_\beta(r_3) = z_\beta(r_1) + z_\beta(r_2)$. By Lemma 2 we know that $z_\beta(r_1) > 0$, $z_\beta(r_2) > 0$ and $z_\beta(r_3) > 0$. Thus we deduce $z_\beta(r_3) > z_\beta(r_1)$ and $z_\beta(r_3) > z_\beta(r_2)$. Consequently, any instantiation of the objective function as defined in Definition 9 favors minimal regions. □

Both objective functions presented, i.e.z_1 and $z_{\overline{L}}$, are expressible in terms of the more general objective function z_β as presented in Definition 9. As we have seen the two objective functions may favors different regions. Combining an objective function with the ILP-formulation presented in Definition 8 establishes means to find Petri net places. However, solving one ILP only yields one solution and hence we need means to find a set of places, which together form a Petri net that represents the input event log L.

4.2 Finding Several Places

The most apparent technique to find multiple places is the use of causal relations within an event log. Within the context of this paper we define a causal relation as follows.

Definition 10 (Causal relation). *Let L be an event log over a set of activities A. A causal relation γ_L is a function $\gamma_L : A \times A \to \mathbb{R}$ where $\gamma_L(a, b)$ denotes the causality between activity a and b exhibited in L.*

Several approaches exist to compute causalities hence we refer to [6] for an overview of the use of causalities within different process discovery approaches. As a consequence, $\gamma_L(a, b)$ can have different meanings w.r.t. the causality between a and b. Some approaches limit $rng(\gamma_L)$ to $\{0, 1\}$ where $\gamma_L(a, b) = 1$ means that there is a causal relation between a and b whereas $\gamma_L((a, b)) = 0$ means there is no causal relation from a to b. Other approaches map $rng(\gamma_L)$ to the real-valued domain $(-1, 1)$ where a high positive $\gamma_L(a, b)$ value (close to 1) indicates a strong causal relation from a to b and a low negative value (close to -1) indicates a strong causal relation from b to a.

When adopting a causal-based ILP process discovery strategy, we try to find places which will be added in the resulting Petri net, each representing a causality found. A first step is to compute γ_L values given some causal definition. Depending on the actual meaning of the γ_L, whenever we find a causal relation from a to b (possibly because $\gamma_L(a, b) \geq \delta$, where δ is some threshold value), we enrich the constraint body for the given causal constraint as follows:

$$m = 0 \text{ and } \vec{v}(a) = 1 \text{ and } \vec{v}(b) = -1, \text{ if } a, b \in A_s$$

$$m = 0 \text{ and } \vec{x}(a) = 1 \text{ and } \vec{v}(b) = -1, \text{ if } a \in A_d, b \in A_s$$

$$m = 0 \text{ and } \vec{v}(a) = 1 \text{ and } \vec{y}(b) = 1, \text{ if } a \in A_s, b \in A_d$$

$$m = 0 \text{ and } \vec{x}(a) = 1 \text{ and } \vec{y}(b) = 1, \text{ if } a, b \in A_d$$

After the constraint body is enriched, we solve the ILP yielding a place having an optimal value for the specific objective function chosen. We repeat the procedure for every causal relation yielding a Petri net.

5 Performance

The hybrid variable-based ILP-formulation is implemented as a plug-in in the ProM-Framework (`http://www.promtools.org`) [7][2]. The plug-in allows the user to

[2] The source-code is available at `https://svn.win.tue.nl/repos/prom/Packages/HybridILPMiner/Trunk`

specify parameters of ILP-based process discovery, e.g. several different objective functions, additional constraints and causality based mining.

To test the performance of hybrid variable-based ILP process discovery we have used the basic implementation within an experimentation framework[3]. The framework allows for generating *random process models* and from these models generate *event logs*. We have generated 40 models containing a minimum of 2 activities and a maximum of 12 activities. For each model we have generated 10 event logs, each containing 5000 traces. For each log we ran three different instantiations of the process discovery formulation, one purely single variable-based, one hybrid variant and one purely dual variable-based. The distribution of event classes in A over A_s and A_d is depicted in Table 2. For the hybrid variant the size of A_s was kept constant and independent of the number of activities in A (as of $|A| \geq 3$). We ran each instantiation 10 times per log using causal relations as a process discovery strategy. For each run of an instantiation we calculated the total time spend in solving all ILPs based on the causalities present in the event log. These times have been aggregated based on the number of activities present in an event log. The results of the experiment, plotted on a logarithmic scale, are depicted in Figure 3[4].

As shown in Figure 3, the average time to solve the hybrid formulation is in-between the single and dual formulation. We additionally note the hybrid formulation to get slightly closer to the dual formulation when $|A|$ increases. This is as expected as the number of *single* variables within the hybrid formulation is constant and hence the average time of solving the ILPs within the formulation increases with respect to the single variable formulation. Figure 3 shows that reducing the number of variables used within the ILP-formulation has an impact on the average time of solving ILPs. The differences are however marginal, which is to be expected as solving ILPs is exponential by nature. Therefore the experiments show that using ILPs for the aim of process discovery in

Table 2: Different settings used in performance measurements of the hybrid variable-based ILP-formulation.

$\|A\|$	2	3	4	5	6	7	8	9	10	11	12
Purely single variable-based variant											
$\|A_s\|$	2	3	4	5	6	7	8	9	10	11	12
$\|A_d\|$	0	0	0	0	0	0	0	0	0	0	0
Hybrid variable-based variant											
$\|A_s\|$	2	3	3	3	3	3	3	3	3	3	3
$\|A_d\|$	0	0	1	2	3	4	5	6	7	8	9
Purely dual variable-based variant											
$\|A_s\|$	0	0	0	0	0	0	0	0	0	0	0
$\|A_d\|$	2	3	4	5	6	7	8	9	10	11	12

[3] All framework files and results can be found at `https://svn.win.tue.nl/repos/prom/Packages/HybridILPMiner/Tags/papers/source_files/hybrid_ilp_runtime_experiments.tar.gz`

[4] The experiments where distributed over four servers (Dell PowerEdge R520, Intel Xeon E5-2407 v2 2.40GHz, 10M Cache, 8 × 8GB RDIMM, 1600MT/s Memory), on each server a total of 10 models and corresponding logs where generated.

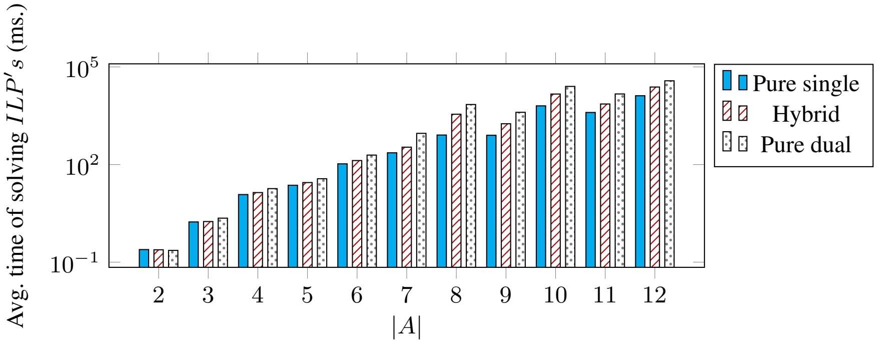


Fig. 3: Average time in milliseconds to solve the three ILP-based process discovery variants plotted on a logarithmic scale against the number of activities in the log.

a practical setting might need incorporation of more advanced techniques in order to reduce the average time of solving the ILPs significantly.

6 Related Work

The concept of hybrid variable-based regions originates from language-based region theory, which in turn originates from the area of Petri net synthesis. We identify two main branches of language-based region theory within Petri net synthesis being the *separating regions approach* [8–10] and the *minimal linear basis approach* [11, 12]. In the separating regions approach, which uses single-variable based regions, behavior not seen in a given prefix closed language is specified as being undesired. In the minimal linear basis approach, given a prefix closed language, a polyhedral cone of integer points is constructed based on dual variable-based regions. Using the cone a minimal basis of the set of regions is calculated which defines a minimal set of places to be synthesized. Both approaches try to minimize $\mathcal{L}(N) \setminus \overline{L}$, where N represents the resulting Petri net and $\overline{L}$ is a prefix-closed language. The approaches lead to Petri nets with perfect replay-fitness. Moreover precision is *maximized*. A side effect of this property is the fact that the synthesized net N scores low on both the generalization and the simplicity dimension.

In [3] a first design of an ILP-formulation was presented based on the concept of dual variable-based regions. The work presents objective function z_1 which we have further developed in this work to $z_{\overline{L}}$, and more generally, z_β. The work also focuses on formulation of several different net-types in terms of linear in-equalities which even go beyond classical Petri nets, e.g. reset- and inhibitor arcs.

An alternative approach is to use the concept of state-based regions for the purpose of process discovery [13, 14]. Within this approach an abstraction of the event log is computed in the form of a *transition system*. Regions are computed within the transition system where, like in language-based region theory, each region corresponds to a place in the resulting Petri net.

In [15] a process discovery algorithm is proposed based on the concept of *numerical abstract domains* using Parikh vectors as a basis. Based on an event log, a prefix-closed language is computed of which a convex polyhedron is approximated by means of calculating a convex hull. The convex hull is then used to compute causalities within the input log, by deducing a set of linear inequalities which represent places. The formulation used to calculate these causalities is in essence based the concept of single variable-based regions. The approach allows for finding pure Petri nets with arc weights and multiple marked places.

A multitude of other Petri net-based process discovery approaches exist. For a detailed description of these approaches we refer to [6].

7 Conclusion

We presented a new breed of language-based regions, i.e. hybrid variable-based regions, that captures the two existing region types being single and dual variable-based regions. Hybrid variable-based regions allow us to decide whether we want to use one or two variables for an activity present in the input event log. This allows us to achieve gains in terms of performance whilst maintaining the possibility to find complex (workflow) patterns. We have shown that within the hybrid variable-based ILP process discovery formulation, using only one variable per activity $a \in A$ performs optimal in terms of the average time spent in solving the ILPs constructed.

We presented a generalized weighted objective function and showed that any instantiation of the objective function leads to ILPs that favor minimal regions. As a result, practitioners may vary the scaling function within the objective function under the guarantee that the objective function favors minimal regions. We presented a log-based scaling function that exploits trace frequencies available in the input log. Using the log based scaling function within the objective function assigns an objective value to each region equal to the token throughput of the corresponding place, given the event log. Hence, as we have shown using the new objective function leads to more intuitive objective values.

As an interesting direction for future work we identify the assessment of the impact of filtering, either a-priori or within the ILP itself, on the quality dimensions of the resulting nets, i.e. are we able to leverage the perfect repaly-fitness property? Also, an assessment of the impact of decomposition techniques [16] on the performance of ILP-based approaches is an interesting direction for future work.

References

1. Aalst, W.M.P.v.d.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. 1st edn. Springer Publishing Company, Incorporated (2011)
2. Desel, J., Reisig, W.: The synthesis problem of Petri nets. Acta Inf. **33**(4) (1996) 297–315
3. Werf, J.M.E.M.v.d., Dongen, B.F.v., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. Fundamenta Informaticae **94**(3) (2009) 387–412
4. Buijs, J.C.A.M., Dongen, B.F.v., Aalst, W.M.P.v.d.: On the role of fitness, precision, generalization and simplicity in process discovery. In Meersman, R., Panetto, H., Dillon, T.,

Rinderle-Ma, S., Dadam, P., Zhou, X., Pearson, S., Ferscha, A., Bergamaschi, S., Cruz, I.F., eds.: On the Move to Meaningful Internet Systems: OTM 2012. Volume 7565 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2012) 305–322

5. Aalst, W.M.P.v.d., Hofstede, A.H.M.t., Kiepuszewski, B., Barros, A.: Workflow patterns. Distributed and Parallel Databases **14**(1) (2003) 5–51
6. Dongen, B.F.v., Medeiros, A.K.A.d., Wen, L.: Process mining: Overview and outlook of Petri net discovery algorithms. T. Petri Nets and Other Models of Concurrency **2** (2009) 225–242
7. Dongen, B.F.v., Medeiros, A.K.A.d., Verbeek, H.M.W., Weijters, A.J.M.M., Aalst, W.M.P.v.d.: The ProM framework: A new era in process mining tool support. In Ciardo, G., Darondeau, P., eds.: Applications and Theory of Petri Nets 2005. Volume 3536 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2005) 444–454
8. Badouel, E., Bernardinello, L., Darondeau, P.: Polynomial algorithms for the synthesis of bounded nets. In Mosses, P.D., Nielsen, M., Schwartzbach, M.I., eds.: TAPSOFT '95: Theory and Practice of Software Development. Volume 915 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (1995) 364–378
9. Badouel, E., Darondeau, P.: Theory of regions. In Reisig, W., Rozenberg, G., eds.: Lectures on Petri Nets I: Basic Models. Volume 1491 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (1998) 529–586
10. Darondeau, P.: Deriving unbounded Petri nets from formal languages. In Sangiorgi, D., Simone, R.d., eds.: CONCUR'98 Concurrency Theory. Volume 1466 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (1998) 533–548
11. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process mining based on regions of languages. In Alonso, G., Dadam, P., Rosemann, M., eds.: Business Process Management. Volume 4714 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2007) 375–383
12. Lorenz, R., Mauser, S., Juhas, G.: How to synthesize nets from languages - a survey. In: Simulation Conference, 2007 Winter. (Dec 2007) 637–647
13. Solé, M., Carmona, J.: Process mining from a basis of state regions. In: Applications and Theory of Petri Nets, 31st International Conference, PETRI NETS 2010, Braga, Portugal, June 21-25, 2010. Proceedings. (2010) 226–245
14. Aalst, W.M.P.v.d., Rubin, V., Verbeek, H.M.W., Dongen, B.F.v., Kindler, E., Günther, C.W.: Process mining: a two-step approach to balance between underfitting and overfitting. Software and System Modeling **9**(1) (2010) 87–111
15. Carmona, J., Cortadella, J.: Process discovery algorithms using numerical abstract domains. IEEE Trans. Knowl. Data Eng. **26**(12) (2014) 3064–3076
16. Aalst, W.M.P.v.d.: Decomposing Petri nets for process mining: A generic approach. Distributed and Parallel Databases **31**(4) (2013) 471–507

Folding Example Runs to a Workflow Net

Robin Bergenthum, Thomas Irgang, Benjamin Meis

Department of Software Engineering,
FernUniversität in Hagen
{firstname.lastname}@fernuni-hagen.de

Abstract. We present a folding algorithm to construct a business process model from a specification. The process model is a workflow net, i.e. a Petri net with explicit split- and join-transitions and the specification is a set of example runs. Each example run is a labeled partial order of events, and each event relates to the occurrence of an activity of the underlying business process. In contrast to sequentially ordered runs, a partially ordered run includes information about dependencies and independencies of events. Consequently, such a run is a precise and intuitive specification of an execution of a business process [5, 11].

The folding algorithm is based on the algorithm introduced in [1]. This algorithm constructs a process model which is able to execute all example runs of the specification, but may introduce a significant amount of not specified behavior to the business process model. We show how to improve this folding procedure, by adapting ideas known from the theory of regions, in order to restrict additional and not specified behavior of the process model whenever possible.

1 Introduction

Business process management [2–4] aims to identify, supervise and improve business processes within companies. It is essential to adapt existing processes to rapidly changing requirements in order to increase and guarantee corporate success. The basis for every business process management activity is a valid and faithful model of the business process; yet constructing such a model is a challenge.

It is particularly challenging to build a complex process model from scratch. For most applications it is easier to first explore single example runs and set up a formal specification before building a complex model [5–7]. In the literature there are many different approaches to automatically generating a process model from a specification. According to the requirements, several approaches exist using different types of specifications, process modeling languages, and model generation strategies.

Process mining algorithms (see [8] for an overview) provide very good runtime, construct readable models, and take into account that recorded or specified behavior can be incomplete or even faulty. Synthesis algorithms (see for example [9–11]) assume a complete and valid specification and construct a process model representing the specification as precisely as possible. Synthesis algorithms are only applicable for medium-size models, but provide excellent control of the produced model and its behavior. In this paper, we present a process mining algorithm and use strategies common in the area of synthesis to improve the construction of the process model. We will show that our new algorithm is fast and able to provide perfect control of the constructed model.

We consider a specification to be a set of labeled partial orders. A labeled partial order is a partially ordered set of events. An event and its label relate to the occurrence of an activity in the business process. In contrast to a sequence of events, a partial order can express dependencies and independencies of events. A set of labeled partial orders is a precise and intuitive specification of a business process [5, 11].

As an example we consider our coffee brewing process. Figure 1 and Figure 2 depict two labeled partial orders (we omit transitive arcs) specifying two different runs of this process. In Figure 1, we grind beans and switch off the coffee machine. We unlock the machine once it is turned off. We fill the strainer as soon as it is empty. We fetch water from the kitchen using the coffee-pot. This pot is only available after the machine is unlocked. Once the strainer is filled and the water is fetched, we assemble the coffee machine. In Figure 2, we use a glass-pot (instead of the coffee-pot) to fetch water from the kitchen. This activity does not depend on unlocking the coffee machine. We can fetch the water right at the beginning of the process. Figure 1 and Figure 2 depict a complete and intuitive specification of our coffee brewing process.

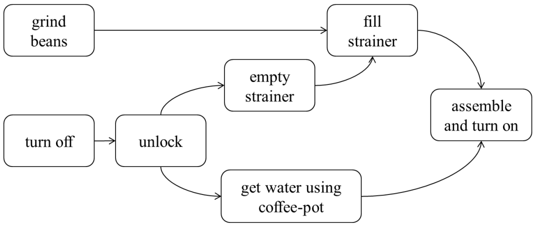


Fig. 1. A Coffee brewing process.

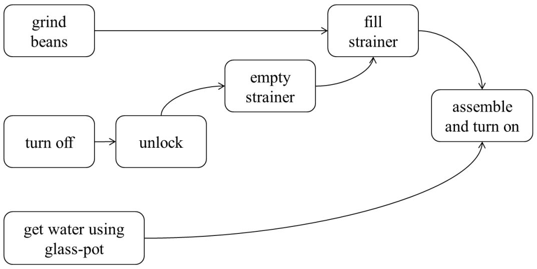


Fig. 2. Another coffee brewing process.

We present an algorithm to construct a workflow net from a specification. As stated above, a specification is a set of labeled partial orders. A workflow net is a Petri net with explicit split- and join-connectors. *and*-splits and *and*-joins duplicate and merge the control flow of a workflow net if actions of the business process occur concurrently.

xor-splits and xor-joins act like switches and steer the course of the control flow of a workflow net if activities of the business process are in conflict.

The new algorithm is based on the folding algorithm presented in [1]. This algorithm adds an initial and a final event to every labeled partial order of the specification and builds a workflow net so that the specification is enabled in this net, i.e. all step sequences of the labeled partial orders of the specification are enabled. The unique start and final events are only necessary for technical purposes and are removed after the folding which leads to a workflow net with a unique start and final place. To build such a net, every labeled partial order is reduced to its underlying Hasse-diagram. A Hasse-diagram of a labeled partial order is the set of events together with the smallest relation so that its transitive closure equals the original partial order. In other words, all transitive arcs are removed to receive a compact and easy to handle representation of the specified example run. We use the Hasse-diagrams of the specification to analyze the neighborhood relation on activities of the business process. For every activity there is a set of equally labeled events. According to the specification, each of these events is enabled by the set of events in its direct preset. Of course, equally labeled events can occur in different contexts. Folding is to arrange actions using splits and joins according to the neighborhood relation present in the Hasse-diagrams of the specification. An xor-split ($\times$) marks exactly one place of its postset, an xor-join ($\times$) needs only one marked place in its preset to be enabled. and-connectors ($\wedge$) use the common Petri net transition semantics.

Folding the specification depicted in Figure 1 and Figure 2 results in the workflow net depicted in Figure 3. Both labeled partial orders of the specification are enabled in this net. For simplification, we omit places between transitions.

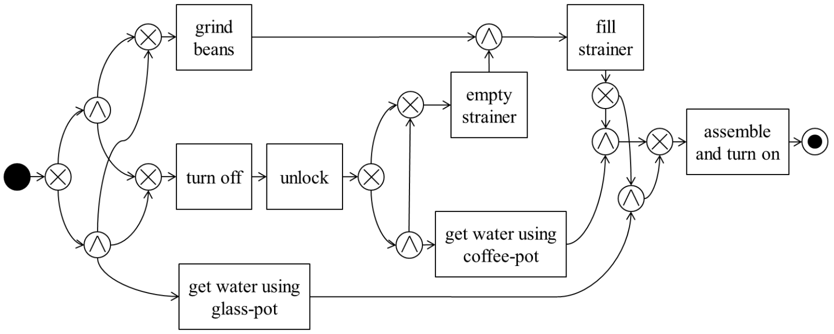


Fig. 3. A business process model of our coffee brewing process.

Folding labeled partial orders is an elegant approach to generating a business process model from a specification. Folding constructs well readable results in very good runtime. Every labeled partial order of the specification is enabled in the generated process model. Unfortunately, folding algorithms tend to introduce additional, not specified behavior to the business process model. Such additional behavior is either a suitable completion of the specification or an inadequate extension yielding an inaccurate business process model. The risk of creating an unfaithful model is due to the fact that the basis for folding is the neighborhood relation introduced by the Hasse-diagrams. Some of the causal structure of the business process may be hidden in the transitive closure of the specified example runs. Transitive dependencies are not considered by the folding algorithm.

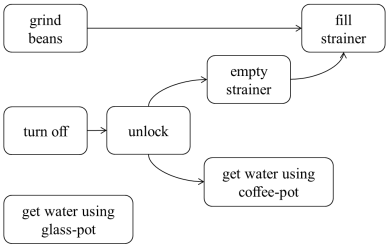


Fig. 4. A run of the business process model depicted in Figure 3.

Figure 4 illustrates additional behavior enabled in the workflow net depicted in Figure 3. In compliance with the diagram in Figure 1, we grind beans, turn the machine off, and get water using the glass-pot right at the beginning. We also unlock the coffee machine and fetch water using the coffee-pot. This behavior is possible in the workflow net, because the alternative between using the glass- and coffee-pot is not reflected by any neighborhood relation of the diagrams depicted in Figure 1 and Figure 2.

In this paper we present a revised folding algorithm. We fold the Hasse-diagrams of the specification into a workflow net. If the workflow net contains additional and inadequate behavior, the revised folding algorithm proceeds to exclude this behavior by changing the workflow net. Nevertheless, all changes lead to a new model which is still able to perform the specified behavior. To improve the model, we use methods known from the theory of synthesis. Each non-specified run of a workflow net has a maximal, specified (not necessarily unique) prefix. Any event ordered after this specified part should not occur at this point. Such specified prefix together with such an event is called a wrong continuation of the workflow net. To eliminate such a wrong continuation from our workflow net, we modify the Hasse-diagrams of our specification. We add a set of transitive arcs to the Hasse-diagrams, so that folding regarding these updated diagrams leads to a workflow net without the wrong continuation. Note, we only add arcs of the transitive closure of the Hasse-diagrams. Thereby, the initial specified behavior is not changed. A more detailed neighborhood relation yields a more restrictive workflow net. We stop if we can not get a more restrictive workflow net by adding transitive arcs. This solution is not unique. It depends on the unfolding and the selected arcs.

We will present the theory and implementation of this revised folding algorithm and we will show that it constructs well readable models. The main advantage of such a folding procedure is that it provides good control of the behavior of the constructed business process model. In an interactive version of our algorithm, it is even possible to distinguish two sets of wrong continuations. The first set is not specified but valid process behavior and extends the specification, the second set is excluded from the model.

The paper is organized as follows: Section 2 defines workflow nets and labeled partial orders. Section 3 presents a folding algorithm. Section 4 presents our new revised folding algorithm and outlines its implementation. Section 5 concludes the paper.

2 Workflow Nets and Labeled Partial Orders

In this paper, a workflow net is a Petri net with additional connector nodes. We consider a special class of these nets where transitions and places do not branch. This class of nets has a very intuitive semantics and is built with elements present in almost every other process modeling language. A workflow net can easily be translated into any business process modeling language such as place/transition nets [12], Event-driven Process Chains (EPCs) [13], Business Process Model and Notation (BPMN) [14], Yet Another Workflow Language (YAWL) [15] or Activity Diagrams (a part of the Unified Modeling Language (UML) [16, 17]).

Definition 1. *A workflow net structure is a tuple* $wn = (T, P, C_{xor}, C_{and}, F)$ *where* T *is a finite set of transitions,* P *is a finite set of places,* C_{xor} *resp.* C_{and} *are finite sets of* xor*- resp.* and*-connectors, and* $F \subseteq ((T \cup C_{xor} \cup C_{and}) \times P) \cup (P \times (T \cup C_{xor} \cup C_{and}))$ *is a set of directed arcs connecting transitions and connectors to places and vice versa. A workflow net structure is a workflow net if:*

(i) There is one place, called initial place, having one outgoing and no incoming arc. There is one place, called final place, having one incoming and no outgoing arc. All other places have one incoming and one outgoing arc.
(ii) Transitions have one incoming and one outgoing arc.
(iii) Connectors have either one incoming and multiple outgoing arcs, or multiple incoming and one outgoing arc.

Figure 3 depicts a workflow net. For the sake of clarity, places are hidden in this figure. Let $n \in \{T \cup C_{xor} \cup C_{and}\}$ be a node. We call $\bullet n := \{p \in P \mid (p, n) \in F\}$ the preset of n. We call $n\bullet := \{p \in P \mid (n, p) \in F\}$ the postset of n. A marking of a workflow net assigns tokens to places.

Definition 2. *Let* $w = (T, P, C_{xor}, C_{and}, F)$ *be a workflow net. A marking of* w *is a function* $m : P \to \mathbb{N}_0$*. A pair* (w, m) *is called marked workflow net. A place* $p \in P$ *is called marked if* $m(p) > 0$*, marked by one if* $m(p) = 1$ *and unmarked if* $m(p) = 0$ *holds. The initial marking* m_0 *of a workflow net is defined as follows: The initial place is marked by one and all other places are unmarked.*

There is a simple firing rule for workflow nets. A node is enabled to fire if every place in its preset is marked. An xor-join is also enabled if there is at least one marked place in its preset.

Definition 3. *Let* $wn = (T, P, C_{xor}, C_{and}, F, m)$ *be a marked workflow net. A node* $n \in \{T \cup C_{and}\}$ *is enabled if every place in* $\bullet n$ *is marked. A node* $n \in C_{xor}$ *is enabled* *if there is a marked place in $\bullet n$. If a node is enabled, it can fire, changing the marking of the workflow net. Firing $n \in (T \cup C_{and})$ leads to the marking m' defined by:*

$$m'(p) = \begin{cases} m(p) - 1, p \in \bullet n \setminus n\bullet \\ m(p) + 1, p \in n \bullet \setminus \bullet\, n \\ m(p) \qquad else. \end{cases}$$

If a node $n \in C_{xor}$ is enabled, choose a marked place $p_{in} \in \bullet n$ and a place $p_{out} \in n\bullet$. Firing n leads to the marking m' defined by:

$$m'(p) = \begin{cases} m(p) - 1, p_{in} = p \neq p_{out} \\ m(p) + 1, p_{out} = p \neq p_{in} \\ m(p) \qquad else. \end{cases}$$

If an enabled node n fires and changes m to m', we write $m[n\rangle m'$.

A set of nodes is called a step. In a workflow net there are no conflicts regarding the consumption of tokens. Consequently, a step is enabled if each node of the step is enabled. Firing a step leads to the same marking as firing all nodes.

Definition 4. *Let $N \subseteq \{T \cup C_{xor} \cup C_{and}\}$ be a step and m be a marking. N is enabled in m if each $n \in N$ is enabled in m. If an enabled step N fires and changes m to m', we write $m[N\rangle m'$. Firing $N = \{n_1, \ldots, n_n\}$ leads to the same marking as firing all $n \in N$, i.e. $m[n_1\rangle m_1[n_2\rangle \ldots [n_n\rangle m'$.*

Let $\sigma = N_1\, N_2\, \ldots\, N_n$ be a sequence of steps. The sequence σ is enabled in m if there are $m_1, m_2, \ldots, m_n$, so that $m[N_1\rangle m_1[N_2\rangle \ldots [N_n\rangle m_n$ holds. If σ is enabled, we define $\sigma_T^{\emptyset} = (N_1 \cap T)\,(N_2 \cap T)\, \ldots\, (N_n \cap T)$. We omit all empty sets in $\sigma_T^{\emptyset}$ to define σ_T. We call σ_T the transition step sequence of σ.

We call a step N a transition step if $N \subseteq T$. A sequence of transition steps τ is enabled in m if there is an enabled sequence of steps σ' so that $\tau = \sigma'_T$ holds.

In Figure 3, the transition step sequence $\{$*grind beans*, *turn off*$\}\{$*unlock*$\}\{$*empty strainer*, *get water using coffee-pot*$\}\{$*fill strainer*$\}\{$*assemble and turn on*$\}$ is enabled in the initial marking. We use transition step sequences to define enabled labeled partial orders [18, 19].

Definition 5. *Let T be a set of labels, a labeled partial order is a triple $lpo = (V, <, l)$, where V is a finite set of events, $<$ is an irreflexive and transitive binary relation over V, and $l : V \to T$ is a labeling function. We consider labeled partial orders without autoconcurrency, i.e. $e, e' \in V, e \neq e', e \not< e', e' \not< e \Rightarrow l(e) \neq l(e')$.*

The Hasse-diagram of a labeled partial order is $lpo^{\lhd} = (V, \lhd, l)$, where $\lhd$ is the set of skeleton arcs, i.e. $\lhd = \{(v, v') \mid v < v' \wedge \nexists v'' : v < v'' < v'\}$.

Let $lpo = (V, <, l)$ and $lpo' = (V, <', l)$ be labeled partial orders. If $< \subseteq <'$ holds, lpo' is a sequentialisation of lpo. If $V = V_1 \,\dot\cup \ldots \dot\cup\, V_n$ and $<' = \bigcup_{i<j} V_i \times V_j$ hold, we call the sequence $l(V_1)\, \ldots\, l(V_n)$ a transition step sequence of lpo.

The sequence $\{$*grind beans*, *turn off*$\}\{$*unlock*$\}\{$*empty strainer*, *get water using coffee-pot*$\}\{$*fill strainer*$\}\{$*assemble and turn on*$\}$ is a transition step sequence of the labeled partial order of Figure 1.

Definition 6. *Let wn be a marked workflow net. A labeled partial order lpo is enabled in wn if all transition step sequences of lpo are enabled in the initial marking of wn.*

Figure 1, Figure 2, and Figure 4 depict Hasse-diagrams of labeled partial orders enabled in the initial marking of the workflow net depicted in Figure 3.

3 Folding Algorithm

We introduce a folding algorithm to construct a workflow net from a specification which is a set of labeled partial orders. Each partial order corresponds to a run of the business process. Events model occurrences of activities, arcs model dependencies between events, and unordered events can occur concurrently. Our revised folding algorithm is based on the folding algorithm presented in [1]. We add an initial and a final event to every labeled partial order, before reducing every order to its Hasse-diagram. From these we deduce a neighborhood relation on the set of labels. We define a set of preceding and succeeding label sets for each label. Every label of the specification can, of course, occur multiple times, even in one labeled partial order.

Definition 7. *Let $lpo = (V, <, l)$ be a labeled partial order, let $(V, \lhd, l)$ be its Hasse-diagram, and let T be a set of labels with $l(V) \subset T$. Let $e \in V$ be an event, denote $pred(e) = \{l(e')|e' \lhd e\}$ the set of preceding labels, and denote $succ(e) = \{l(e')|e \lhd e'\}$ the set of succeeding labels.*

Let L be a set of labeled partial orders. Let $t \in T$ be a label, denote $predset(t) = \{pred(e)|(V, <, l) \in L, e \in V, l(e) = t\}$ the set of preceding label sets, and denote $succset(t) = \{succ(e)|(V, <, l) \in L, e \in V, l(e) = t\}$ the set of succeeding label sets.

To construct a workflow net from a specification, we construct a transition for every label and connect transitions according to the corresponding preceding and succeeding label sets. For every transition we will define a so called building block. The center of each building block is the transition, surrounded by three layers of connectors. Next to the transition is a layer of two *xor*-connectors, because each transition can have multiple preceding and multiple succeeding label sets. For each of these sets, there is an *and*-connector on the second layer, because each set can have multiple labels. If succeeding or preceding label sets share labels, there is a *xor*-connector on the third layer. We define a building block as follows:

Definition 8. *Let L be a set of labeled partial orders and T be its set of labels. For each label $t \in T$ we define a workflow net structure $wn^t = (\{t\}, P^t, C^t_{xor}, C^t_{and}, F^t)$ called building block of t. The sets P^t, C^t_{xor}, and C^t_{and} are defined as follows:*

$$\begin{aligned} C^t_{xor} = \{&xor^t_{pre}, xor^t_{post}\} \cup \{xor^t_{pre,t'}|t' \in X, X \in predset(t)\} \cup \\ &\{xor^t_{post,t'}|t' \in X, X \in succset(t)\}, \end{aligned}$$

$$C^t_{and} = \{and^t_{pre,X}|X \in predset(t)\} \cup \{and^t_{post,X}|X \in succset(t)\},$$

$$\begin{aligned} P^t = \{&p^t_{pre}, p^t_{post}\} \cup \\ &\{p^t_{pre,X}|X \in predset(t)\} \cup \{p^t_{post,X}|X \in succset(t)\} \cup \\ &\{p^t_{pre,t',X}|t' \in X, X \in predset(t)\} \cup \{p^t_{post,X,t'}|t' \in X, X \in succset(t)\} \cup \\ &\{p^t_{pre,t'}|t' \in X, X \in predset(t)\} \cup \{p^t_{post,t'}|t' \in X, X \in succset(t)\}. \end{aligned}$$

The set of arcs F^t is defined as follows:

$$\begin{aligned}
F^t = \{&(xor^t_{pre}, p^t_{pre}), (p^t_{pre}, t), (t, p^t_{post}), (p^t_{post}, xor^t_{post})\} \cup \\
&\{(and^t_{pre,X}, p^t_{pre,X}) | X \in predset(t)\} \cup \\
&\{(p^t_{pre,X}, xor^t_{pre}) | X \in predset(t)\} \cup \\
&\{(xor^t_{post}, p^t_{post,X}) | X \in succset(t)\} \cup \\
&\{(p^t_{post,X}, and^t_{post,X} | X \in succset(t)\} \cup \\
&\{(xor^t_{pre,t'}, p^t_{pre,t',X}) | t' \in X, X \in predset(t)\} \cup \\
&\{(p^t_{pre,t',X}, and^t_{pre,X}) | t' \in X, X \in predset(t)\} \cup \\
&\{(and^t_{post,X}, p^t_{post,X,t'}) | t' \in X, X \in succset(t)\} \cup \\
&\{(p^t_{post,X,t'}, xor^t_{post,t'}) | t' \in X, X \in succset(t)\} \cup \\
&\{(p^t_{pre,t'}, xor^t_{pre,t'}) | t' \in X, X \in predset(t)\} \cup \\
&\{(xor^t_{post,t'}, p^t_{post,t'}) | t' \in X, X \in succset(t)\}.
\end{aligned}$$

Figure 5 depicts the building block of label *unlock*. There are two events labeled by *unlock* in Figure 1 and Figure 2. Both events have the same set of preceding labels, i.e. $\{$*turn off*$\}$, but have different sets of succeeding labels. According to these sets there is one *xor*-connector and two *and*-connectors right behind transition *unlock*. Since both sets share the label *empty strainer*, the control flow is joined with $xor^{unlock}_{post,\ empty\ strainer}$ in front of the outgoing interface place $p^{unlock}_{post,\ empty\ strainer}$.

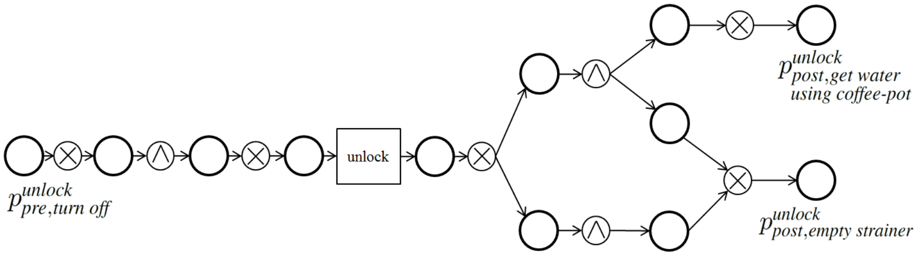

unlock
$p^{unlock}_{pre,turn\ off}$
$p^{unlock}_{post,get\ water\ using\ coffee\text{-}pot}$
$p^{unlock}_{post,empty\ strainer}$

Fig. 5. The building block of label unlock.

We call a building block a compressed building block if all superfluous connectors and the corresponding places are removed. A connector is superfluous if it has one ingoing and one outgoing arc. As an example, Figure 6 depicts the set of compressed building blocks of our coffee brewing process. Note that, before building these blocks, an event labeled with start, and an event labeled with stop, are added to every labeled partial order. In this figure, we depict transition *start* by a big black dot, transition *stop* by a circle with a dot. We hide most places and only sketch interface places by small dots labeled by the corresponding preceding or succeeding labels. On the top left of Figure 6, we depict the building block of label *start*. Next to this block, there is the compressed version of the building block depicted in Figure 5. We merge all compressed building blocks depicted in Figure 6 to get the workflow net depicted in Figure 3.

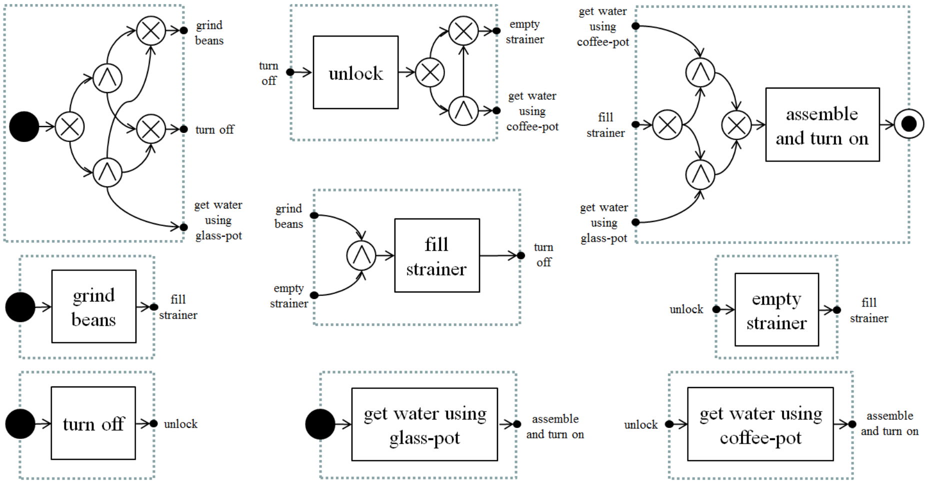


Fig. 6. Compressed building blocks.

Algorithm 1 Folding

```
1: **input**: Specification $L$
2: $L \leftarrow$ add an initial event labeled $start$ to every $lpo \in L$
3: $L \leftarrow$ add a final event labeled $stop$ to every $lpo \in L$
4: $H \leftarrow$ Hasse-diagrams of $L$
5: $T \leftarrow$ Labels of $L$
6: $B \leftarrow$ Building blocks of $T$
7: $wn \leftarrow$ Merge all building blocks at matching interface places
8: $wn \leftarrow$ Delete superfluous connectors from $wn$ by merging the preset and postset places
9: $wn \leftarrow$ Remove place $p_{pre}^{start}$ and transition $start$
10: $wn \leftarrow$ Mark $p_{post}^{start}$ by one token
11: $wn \leftarrow$ Remove place $p_{post}^{stop}$ and transition $stop$
12: **return** $wn$
```

Algorithm 1 implements the folding procedure. The input is a set of labeled partial orders. In Line 2 and Line 3 we add two additional events, one labeled *start* and one labeled *stop*, to every labeled partial order. We extend each partial order so that the *start* events are earlier than every other event of their partial order and the *stop* events are later than every other event of their partial order. These new events will result in two additional building blocks responsible for starting and ending runs of the workflow net model. In Line 4, we reduce the specifications to Hasse-diagrams and collect the set of labels (including *start* and *stop*). In Line 6, according to Definition 8, we build a building block for every label and in Line 7, we merge all building blocks at matching interface places, i.e. for every pair of labels we merge all places $p_{post,t'}^{t}$ and $p_{pre,t}^{t'}$. In Line 8, we delete superfluous connectors. In addition, we remove the xor-connector in front of transition *start* and the xor-connector right behind transition *stop*. We remove connectors by merging preset and postset places and we also delete corresponding arcs.

In Line 9, we delete transition *start* and place p_{pre}^{start} in its preset. Thereby, place p_{pre}^{start} becomes the initial place of the workflow net and we mark this place by one token. In Line 11, we remove transition *stop* and place p_{post}^{stop} in its preset to get the final result of the folding procedure. Altogether, Algorithm 1 constructs a workflow net enabled to execute every labeled partial order of the specification. For the proof we refer the reader to [1] but state the following theorem.

Theorem 1. *Let L be a set of labeled partial orders and construct a workflow net wn from L using Algorithm 1. Each labeled partial order $lpo \in L$ is enabled in wn.*

4 Revised Folding Algorithm

In this section we will introduce a revised folding algorithm to construct a workflow net from a specification. Folding is very efficient and generates an intuitive workflow net, but for most examples the workflow net is able to execute additional runs. This is reasonable if the specification is incomplete. However, if we assume that the specification is complete, additional behavior should not be included in the business process model. In the following, we detect and deal with additional behavior introduced within the folding procedure.

During the folding procedure, Hasse-diagrams define sets of preceding and succeeding transitions but sometimes, considering only these dependencies is insufficient. In a business process an early decision can easily determine later alternatives.

We consider Figure 1 and Figure 2 as an example. There is only one event labeled *get water using coffee-pot*. The building block of *get water using coffee-pot* has one preceding label set, i.e. $\{$*unlock*$\}$. According to Figure 2, the transitions *get water using glass-pot*, *turn off*, and *unlock* can occur. This prefix enables *get water using coffee-pot* by the occurrence of *unlock*. This results in the Hasse-diagram depicted in Figure 4.

As stated above, the Hasse-diagrams of the specification define preceding and succeeding label sets to construct building blocks. Considering these sets defined by the transitive relation of the labeled partial orders constructs a workflow net with minimal additional behavior. The occurrence of any transition in this net is conditioned by the occurrence of all transitions corresponding to the complete history of a corresponding event. Obviously, this leads to an unreadable workflow net with a huge number of connectors and arcs.

Our aim is to identify so-called dependency diagrams, a compromise between the Hasse-diagrams and the partial orders, in order to modify the specification thus that additional behavior of a folded model is restricted as far as possible. The fewer dependencies we add, the smaller is the constructed workflow net.

Definition 9. *Let $(V, <, l)$ be a labeled partial order, let $(V, \lhd, l)$ be its Hasse-diagram, and denote T the set of labels. Let (D, E) be a pair of sets of labels, we denote $\lhd^{[D,E]} = \lhd \cup \{(e, e')|e < e', l(e) \in D, l(e') \in E\}$ the dependency relation of $<$ with regard to (D, E). We call $(V, \lhd^{[D,E]}, l)$ the dependency diagram of $(V, <, l)$ with regard to (D, E).*

Of course, $\lhd^{[\emptyset,\emptyset]} = \lhd$ and $\lhd^{[T,T]} =<$, i.e. every dependency diagram is some tradeoff between the Hasse-diagram and the labeled partial order.

Our revised folding algorithm starts by constructing a workflow net from a set of Hasse-diagrams. If this workflow net contains additional behavior, an enabled but not specified partial order is generated. We modify the set of Hasse-diagrams to get a set of dependency diagrams so that folding these diagrams leads to a net that does not enable the additional partial order. We repeat this procedure to get a workflow net that has no additional behavior if such a workflow net exists. Let us now take a closer look at every step of the revised folding algorithm.

We construct the initial workflow net using Algorithm 1. We generate the behavior of this model by an unfolding procedure. We calculate a so-called branching process [21, 20, 22] describing all enabled labeled partial orders. It is easy to calculate such branching processes for workflow nets because they only branch at xor-splits. Moreover, we do not calculate the complete (maybe infinite) behavior of the workflow net but stop generating the behavior as soon as we construct not specified behavior. If there is not specified behavior, there is at least one so-called wrong continuation. A wrong continuation is a labeled partial order enabled in the workflow net and not part of the specification. Removing one event from a wrong continuation yields a specified partial order. Wrong continuations were originally defined in the area of synthesis of Petri nets from step sequences [9] and for synthesizing Petri nets from partial orders [11].

Definition 10. *Let L be a specification and let T be the set of labels. A labeled partial order $(V, <, l) \notin L$ is called a wrong continuation if there is an event $e \in V$ so that $(V \backslash \{e\}, < |_{(V \backslash \{e\}) \times (V \backslash \{e\})}, l|_{V \backslash \{e\}}) \in L$ holds.*

We consider Figure 4 as an example. The events *grind beans*, *turn off*, *unlock*, *get water using coffee-pot*, and *get water using glass-pot* form a wrong continuation.

In the last step of the revised folding algorithm, we modify the specification to exclude a wrong continuation. The main idea is to extend the preceding and succeeding label sets appropriately, before restarting the folding procedure.

Definition 11. *Let $L = \{(V_1, <_1, l_1), \dots, (V_n, <_n, l_n)\}$ be a specification and $L^{\lhd} = \{(V_1, \lhd_1, l_1), \dots, (V_n, \lhd_n, l_n)\}$ be its set of Hasse-diagrams. Denote T the set of labels. Let $(V_w, <_w, l_w)$ be a wrong continuation and $(V_w, \lhd_w, l_w)$ be its Hasse-diagram.*

Let (D, E) be a pair of label sets and let l, l' be two labels. We call l dependent on l' if for all $(V, <, l) \in L, v \in V, l(v) = l$: $l' \in \{l(v') | v' \lhd^{[D,E]} v\}$. We denote $M^{[D,E]}(l')$ the set of all labels that depend on l'.

(D, E) is called disabling pair of $(V_w, <_w, l_w)$ if one of the following conditions holds:

(a) There is an $e' \in V_w$ so that there is no $(V_i, <_i, l_i) \in L, e \in V_i, l_i(e) = l_w(e')$: $\{l_i(v) | v \lhd_i^{[D,E]} e\} \subseteq \{l_w(v) | v <_w e'\}$ holds.

(b) There is an $e' \in V_w$ so that there is no $(V_i, <_i, l_i) \in L, e \in V_i, l_i(e) = l_w(e')$: $\{l_i(v) | e \lhd_i^{[D,E]} v\} \supseteq \{l_w(v) | e' \lhd_w^{[D,E]} v\} \cap M^{[D,E]}(l_w(e'))$ holds.

A disabling pair (D, E) defines a modification of a specification. This modification yields a set of dependency diagrams. Every dependency diagram includes the Hasse-diagram and extends this diagram by all transitive arcs leading from labels in D to labels in E. We denote the resulting specification by $L^{[D,E]}$.

Theorem 2. *Let L be a specification, lpo_w be a wrong continuation, and (D, E) be a disabling pair of lpo_w. If we construct a workflow net wn from $L^{[D,E]}$ using Algorithm 1, lpo_w is not enabled in wn.*

Proof. Either (a) or (b) of Definition 11 holds.

If (a) holds, there is an event $e' \in V_w$ for which no event $e \in V_i$, $(V_i, <_i, l_i) \in L$, $l_i(e) = l_w(e')$ exists so that $\{l_i(v)|v \lhd_i^{[D,E]} e\} \subseteq \{l_w(v)|v <_w e'\}$ holds.

Algorithm 1 will build and-connectors related to preceding label sets in the building block of $l(e')$. For every and-connector there is a choice of $e \in V_i$, $(V_i, <_i, l_i) \in L$, $l_i(e) = l_w(e')$ so that the and-connector is related to $\{l_i(v)|v \lhd_i^{[D,E]} e\}$. $\{l_i(v)|v \lhd_i^{[D,E]} e\}$ is not included in $\{l_w(v)|v <_w e'\}$. After the occurrence of $\{l_w(v)|v <_w e'\}$ the and-connector is not enabled. The same holds for every other preceding and-connector of the building block of $l_w(e')$. e' can not occur after the occurrence of its prefix. lpo_w is not enabled in wn.

If (b) holds, there is an event $e' \in V_w$ for which no event $e \in V_i$, $(V_i, <_i, l_i) \in L$, $l_i(e) = l_w(e')$ exists so that $\{l_i(v)|e \lhd_i^{[D,E]} v\} \supseteq \{l_w(v)|e' \lhd_w^{[D,E]} v\} \cap M^{[D,E]}(l_w(e'))$ holds.

Algorithm 1 will build and-connectors related to succeeding label sets in the building block of $l(e')$. For every and-connector there is a choice of $e \in V_i$, $(V_i, <_i, l_i) \in L$, $l_i(e) = l_w(e')$ so that the and-connector is related to $\{l_i(v)|e \lhd_i^{[D,E]} v\}$. $\{l_w(v)|e' \lhd_w^{[D,E]} v\} \cap M^{[D,E]}(l_w(e'))$ is not included in $\{l_i(v)|e \lhd_i^{[D,E]} v\}$. The occurrence of this and-connector will not enable all actions in $\{l_w(v)|e' \lhd_w^{[D,E]} v\} \cap M^{[D,E]}(l_w(e'))$, but every such action depends on the occurrence of $l(e')$. The same holds for every other and-connector of the building block $l(e')$. When executing lpo_w in wn there is at least one action missing a token from the building block $l(e')$. lpo_w is not enabled in wn.

Both conditions (a) and (b) suppress the executability of a wrong continuation in a workflow net representing the dependencies introduced from the disabling pair. As an example, we consider the wrong continuation depicted in Figure 4. A disabling pair of label sets is $(\{$*start*$\}, \{$*get water using coffee-pot*$\})$. In the Hasse-diagram depicted in Figure 1, the succeeding label set B_1 of $start$ according to this disabling pair is $\{$*grind beans*, *turn off*, *get water using coffee-pot*$\}$. In other words, *get water using coffee-pot* is added to the original succeeding label set. The succeeding label set B_2 of $start$ of Figure 2 stays unchanged. In Figure 4 the succeeding label set W of $start$ according to the disabling pair is $\{$*grind beans*, *turn off*, *get water using glass-pot*, *get water using coffee-pot*$\}$. This set W is not covered by B_1 or B_2 so that condition (b) of Definition 11 holds. The wrong continuation is not enabled if the additional dependency between *start* and *get water using coffee-pot* is considered when constructing corresponding building blocks. Altogether, if we add one transitive arc to the Hasse-diagram depicted in Figure 1 (from *start* to *get water using coffee-pot*) and apply Algorithm 1, we construct a workflow net which is not able to execute the Hasse-diagram depicted in Figure 4. In this example, the resulting workflow net (depicted in Figure 7) behaves exactly as specified.

Algorithm 2 implements the revised folding procedure. The input is a set of labeled partial orders. In Line 3 and Line 4, we invoke Algorithm 1. While the result of Algorithm 1 has additional behavior, we calculate a wrong continuation in Line 6. We

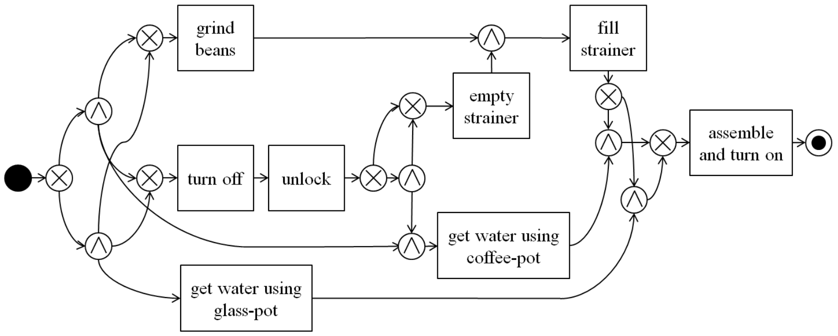


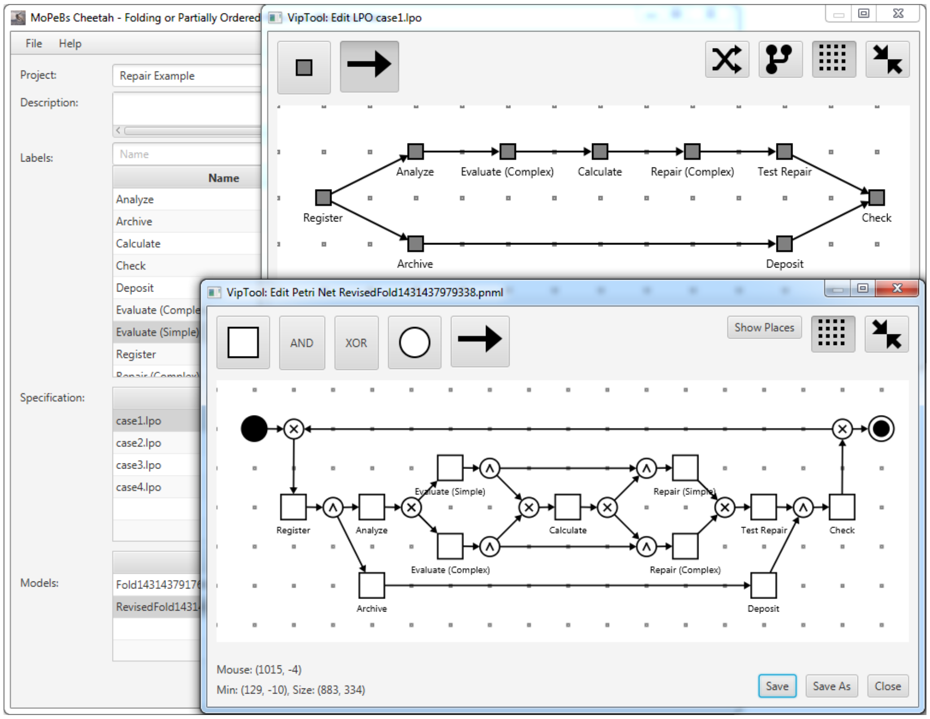
Fig. 7. A business process model of our coffee brewing process.

Algorithm 2 Revised Folding

```
 1: **input**: Specification $L$
 2: $W \leftarrow \emptyset$
 3: $H \leftarrow$ Hasse-diagrams of $L$
 4: $wn \leftarrow$ Folding of $H$
 5: **while** $L(wn) \supset (L \cup W)$ **do**
 6:     $w \leftarrow$ a wrong continuation of $wn$
 7:     **if** there is a disabling pair $(D, E)$ of $w$ **then**
 8:         $H \leftarrow$ expand $H$ by all arcs of $L$ in $(D \times E)$
 9:         $wn \leftarrow$ Folding of $H$
10:     **else**
11:         $W \leftarrow W \cup \{w\}$
12: **return** $wn$
```

construct a disabling pair (if such a pair exists) or add the wrong continuation to a set W. W contains all wrong continuations which cannot be excluded from a workflow net including the specified behavior. In Line 8, we update the set of Hasse-diagrams by constructing a set of dependency diagrams. We fold again to get a new workflow net still including the specified behavior (and W), but excluding the wrong continuations (Line 9). Just like Algorithm 1, Algorithm 2 constructs a workflow net able to execute every labeled partial order of the specification. Furthermore, Algorithm 2 excludes not specified behavior whenever possible.

The runtime of the new algorithm consists of three parts. It is the sum of the runtime of the folding procedures, the runtime of the unfolding procedures (to check for wrong continuations), and the runtime of the calculations of disabling pairs. Every folding procedure is fast. For every event we compute the preceding and succeeding labels and build the corresponding connectors in the workflow net. The worst case complexity of the unfolding procedure is in exponential time. However, the average runtime, where a workflow net has a reasonable level of concurrent activities, is still fast and is determined by the number of xor-split connectors. The most time consuming part is to find a disabling pair. Altogether, the presented algorithm can be slow, especially if the workflow net has a lot of wrong behavior and describes a lot of concurrency. But in this case, it is possible to stop the algorithm after each iteration and still have a reasonable result.



MoPeBs Cheetah - Folding or Partially Ordered

Project: Repair Example

Description:

Labels:

Name
Analyze
Archive
Calculate
Check
Deposit
Evaluate (Comple
Evaluate (Simple)
Register

Specification:

- case1.lpo
- case2.lpo
- case3.lpo
- case4.lpo

Models:

- Fold14314379176
- RevisedFold14314

VipTool: Edit LPO case1.lpo

VipTool: Edit Petri Net RevisedFold1431437979338.pnml

Mouse: (1015, -4)

Min: (129, -10), Size: (883, 334)

Fig. 8. Screenshot of folding results in MoPeBs

The presented revised folding approach is implemented and available in our tool called MoPeBs Cheetah. MoPeBs Cheetah is a lightweight editor showcasing the revised folding algorithm plug-in of our tool set VipTool [23]. VipTool supports various algorithms related to partially ordered behavior of Petri nets [24, 25]. Figure 8 depicts a screenshot of MoPeBs Cheetah. MoPeBs Cheetah (including examples for the folding algorithm and the revised folding algorithm) is available at https://www.fernuni-hagen.de/sttp/forschung/mopebs.shtml.

5 Conclusion

We recapitulated a folding algorithm to generate a workflow net from a specification. The specification is a set of labeled partial orders. The presented algorithm generates an intuitive model by representing the direct dependencies included in the specification. The generated workflow net is able to execute all specified runs. Moreover, this algorithm usually rounds off the specification, i.e. additional runs which are similar to the specified labeled partial orders are executable in the generated workflow net as well. This is reasonable in cases where the specification is considered to be incomplete.

Reusing the folding algorithm, we introduced a revised folding approach. Starting with an initial model, this iterative approach is able to discover transitive dependencies in the specification which yield a more precise process model. The size of the generated model heavily depends on the number of wrong continuations but for most examples, the generated results are readable as well. The generated workflow net can easily be translated into an EPC, BMPN-model, YAWL-model or an Activity Diagram. These are often used for practical applications. Using an interactive version of the revised folding approach, it is easy to validate the specification while generating a process model. We can add reasonable wrong continuations to the specification while excluding unwanted behavior. All in all, in contrast to other process mining algorithms, the revised folding approach provides perfect control over the language of the generated business process model.

References

[1] Bergenthum, R.; Mauser, S.: Folding Partially Ordered Runs. *Proc. of workshop Application of Region Theory (ART) 2011* (Desel, J.; Yakovlev, A. eds.), CEUR 725, 52–62

[2] Mayr, H. C.; Kop, C.; Esberger, D.: Business Process Modeling and Requirements Modeling. *Proc. of International Conference on the Digital Society (ICDS) 2007*, IEEE, Los Alamitos

[3] Oestereich, B.: Objektorientierte Geschäftsprozessmodellierung und modellgetriebene Softwareentwicklung. *HMD-Praxis Wirtschaftsinformatik 241*, dpunkt.verlag, 27–33

[4] Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer, Berlin, 2012

[5] Glinz, M.: Improving the quality of requirements with scenarios. *Proc. of World Congress on Software Quality 2000*, JUSE, Yokohama, 55–60

[6] Mayr, H. C.; Kop, C.: A User Centered Approach to Requirements Modeling. *Proc. of Modellierung 2002*, (Glinz, M.; Müller-Luschnat, G. eds.), LNI P-12, 75–86

[7] Desel, J.: From Human Knowledge to Process Models. *Information Systems and e-Business Technologies 2008*, (Kaschek, R.; Kop, C.; Steinberger, C.; Fliedl, G. eds.) LNBIP 5, 84–95

[8] van der Aalst, W. M. P.; van Dongen, B. F.; Herbst, J.; Maruster, L.; Schimm, G.; Weijters, A. J. M. M.: Workflow Mining: A Survey of Issues and Approaches. *Data & Knowledge Engineering 47(2)*, (Chen, P. P. ed.), Elsevier, 2003, Philadelphia, 237–267

[9] Darondeau, P.: Region Based Synthesis of P/T-Nets and its Potential Applications. *Proc. of Petri Nets 2000*, (Nielsen, M.; Simpson, D. eds.), LNCS 1825, 16–23

[10] Badouel, E.; Darondeau, P.: Theory of regions. *Lectures on Petri Nets I: Basic Models*, (Reisig, W.; Rozenberg G. eds.), LNCS 1491, 529–586

[11] Bergenthum, R.; Desel, J.; Lorenz, R.; Mauser, S.: Synthesis of Petri Nets from Term Based Representations of Infinite Partial Languages. Fundamenta Informaticae (95), 187–217

[12] Reisig, W.: Petrinetze: Modellierungstechnik, Analysemethoden, Fallstudien. *Leitfäden der Informatik*, Vieweg+Teubner 2010

[13] Scheer, A.-W.: ARIS – Vom Geschäftsprozess zum Anwendungssystem. Springer, Berlin, 2002

[14] White, S. A.: Introduction to BPMN. IBM Cooperation, 2004

[15] van der Aalst, W. M. P.; ter Hofstede, A. H. M.: YAWL: Yet Another Workflow Language. (Shasha, D.; Vossen, G. eds.) *Information Systems 30(4)*, Elsevier, 2005, 245–275

[16] International Organization for Standardization: Information technology – Object Management Group Unified Modeling Language – Part 1: Infrastructure, ISO 19505-1:2012, 2012

[17] International Organization for Standardization: Information technology – Object Management Group Unified Modeling Language – Part 2: Superstructure, ISO 19505-2:2012, 2012

[18] Kiehn, A.: On the Interrelation Between Synchronized and Non-Synchronized Behaviour of Petri Nets. (Dassow, J.; Reichel, B. eds.) *Journal of Information Processing and Cybernetics 24(1-2)*, Otto von Guericke Universität, 1988, 3–18

[19] Vogler, W.: Modular Construction and Partial Order Semantics of Petri Nets. LNCS 625, 1992

[20] Goltz, U.; Reisig, W.: Processes of Place/Transition-Nets. (Diaz, J. ed.) *Automata, Languages and Programming*, LNCS 154, 1983, 264–277

[21] Goltz, U.; Reisig, W.: The Non-Sequential Behaviour of Petri Nets. (Meyer, A. R. ed.) *Information and Control 57(2)*, Elsevier 154, 1983, 125–147

[22] Bergenthum, R.; Mauser, S.; Lorenz, R.; Juhás, G.: Unfolding Semantics of Petri Nets Based on Token Flows. *Information and Control 57(2)*, (Niwiński, D.; Son Nguyen, H. eds.), Fundamenta Informaticae 94(3), 2009, 331–360

[23] Desel, J.; Juhás, G.; Lorenz, R.; Neumair, C.:: Modelling and Validation with VipTool. *Proc. of Business Process Management*, (van der Aalst, W. M. P.; Weske, M. eds.), LNCS 2678, 2003, 380–389

[24] Bergenthum, R.; Mauser, S.: Synthesis of Petri Nets from Infinite Partial Languages with VipTool. *Proc. of workshop Algorithmen und Werkzeuge für Petrinetze 2008*, (Lohmann, N.; Wolf, K. eds.), CEUR 380, 2003, 81–86

[25] Bergenthum, R.; Desel, J.; Juhás, G.; Lorenz, R.: Can I Execute My Scenario In Your Net? VipTool Tells You!. *Proc. of Petri Nets and Other Models of Concurrency 2006*, (Donatelli, S.; Thiagarajan, P.S. eds.), LNCS 4024, 2006, 381–390

Mining Duplicate Tasks from Discovered Processes

Borja Vázquez-Barreiros, Manuel Mucientes, and Manuel Lama

Centro de Investigación en Tecnoloxías da Información (CiTIUS)
Universidade de Santiago de Compostela, Santiago de Compostela, Spain
{borja.vazquez,manuel.mucientes,manuel.lama}@usc.es

Abstract. Including duplicate tasks in the mining process is a challenge that hinders the process discovery as algorithms need an extra effort to find out which events of the log belong to which transitions. To face this problem, we propose an approach that uses the local information of the log to enhance an already mined model by performing a local search over the potential tasks to be duplicated. This proposal has been validated over 36 different solutions, improving the final model in 35 out of 36 of the cases.

Keywords: Process mining, process discovery, duplicate tasks.

1 Introduction

The notion of duplicate tasks —or activities— refers to situations in which multiple tasks in the process have the same label. This kind of behavior is useful when i) a particular task is used in different contexts in a process and ii) to enhance the comprehensibility of a model. Typically, duplicate tasks are recorded with the same label in the log and, hence, they hinders the discovery of the model that better fits the log, as algorithms need an extra effort to find out which events of the log belong to which transitions. There are several techniques allowing to mine duplicate tasks [2,3,4,5,6,7], however, or the heuristics rules used to detect the duplicate tasks are not sufficiently general for all the logs [7], or they have to deal with a large search space, increasing the time needed for these algorithms [3,5,6].

In this paper we present a novel proposal to tackle duplicate tasks. The proposal starts from an already mined model without duplicate tasks, and uses the local information of the log and the retrieved process to improve the model through a local search over the potential duplicate tasks.

2 Local search algorithm

Algorithm 1 describes the proposed approach to tackle duplicate tasks. The first step is the discovery of the potential duplicate activities. We used the heuristics defined in [5] to reduce the search space by stating that two tasks with the same

Algorithm 1: Local search Algorithm.

```
   input: A log $L$
1  $ind_0 \leftarrow$ initial_solution(L)                    // Retrieved by a process discovery technique.
2  $potentialDuplicates \leftarrow \emptyset$
3  foreach activity $t$ in the log $L$ do
4      if $\max(\min(|t >_L t'|, |t' >_L t|), 1) > 1$ then
5          $potentialDuplicates \leftarrow potentialDuplicates \cup t$
6  $ind_0 \leftarrow$ localSearch ($ind_0$, $L$, $potentialDuplicates$, true)
7  Function localSearch( $ind_0$, $L$, $potentialDuplicates$, $firstExecution$)
8      $ind_{best} \leftarrow ind_0$
9      $potentialDuplicatesL2L \leftarrow \emptyset$
10     foreach activity $t$ in $potentialDuplicates$ do
11         combinations $\leftarrow$ calculateCombinations ($ind_0$, $L$, t)
12         foreach combination $c$ in combinations do
13             $t' \leftarrow$ activity $t$ from $ind_0$
14             $t.inputs = (t.inputs \setminus c.inputs) \cup c.sharedInputs$
15             $t'.inputs = c.inputs$
16             $t.outputs = (t.outputs \setminus c.outputs) \cup c.sharedOutputs$
17             $t'.outputs = c.outputs$
18             if $(I(t') \neq \emptyset$ && $O(t') \neq \emptyset$ && $I(t) \neq \emptyset$ && $O(t) \neq \emptyset)$ then
19                 Add task $t'$ to individual $ind_0$ and update $t$ in $ind_0$
20                 Repair $ind_0$
21                 Post-prune unused arcs
22                 Evaluate $ind_0$
23                 if $ind_0 < ind_{best}$ then
24                     $ind_0 \leftarrow ind_{best}$
25                 else
26                     $ind_{best} \leftarrow ind_0$
27                     $potentialDuplicatesL2L = potentialDuplicatesL2L \cup t''$ where
                       $t'' \notin potentialDuplicates$ and $t >_L t''$
28             else
29                 $ind_0 \leftarrow ind_{best}$
30     if $firstExecution$ then
31         $ind_{best} \leftarrow$ localSearch ($ind_{best}$, $null$, $potentialDuplicatesL2L$, $false$)
32     return $ind_{best}$
```

label cannot share the same input and output dependencies. Within this context, the duplicate tasks are locally identified based on the *follows relation* ($>_L$), where the upper bound for an activity t is the minimum of the number of tasks that directly precede t in the log and the number of tasks that directly follow t. This definition can be formalized as [5]: $max(min(|t >_L t'|, |t' >_L t|), 1)$. If for a task t the upper bound is greater than 1, then t is considered as a potential task for being duplicated and, hence, it is added to *potentialDuplicates* (Alg.1:3-5).

After finding the potential duplicates, the algorithm splits the input and output dependencies of the activities of the model into multiple tasks with the same label through the function *localSearch* (Alg.1:7). In this step, the algorithm calculates the input and output combinations for each activity in *potentialDuplicates* (Alg. 1:10-11) through the function *CalculateCombinations* (Alg.2). Within this function, the algorithm first finds all the subsequences in the log L that match the pattern t_1tt_2 where $t_1 \in I(t)$ and $t_2 \in O(t)$ in the model (Alg.2:2) —being $I(t)$ and $O(t)$ the inputs and outputs, respectively, of t. Then, based on these

Algorithm 2: Algorithm to compute the combinations of a task.

```
1   Function calculateCombinations(ind, L, t)
        /* If the input parameter $L$ is null, retrieve the sequences from parsing $ind$ */
2       Retrieve all the subsequences $t_1tt_2$ where $t_1 \in I(t)$ and $t_2 \in O(t)$
3       $combinations \leftarrow \emptyset$
4       forall the subsequences $t_1tt_2$ do
5           $c \leftarrow \emptyset$
6           Create a set $c.inputs$ with the combinations that share the same $t_1$ and add in
            $c.outputs$ their respective $t_2$
7           Add $c$ to $combinations$
8       foreach $c$ in $combinations$ do
9           if $c.outputs = c'.outputs$ where $c' \in combinations$ then
10              $c.inputs = c.inputs \cup c'.inputs$ and $c.outputs = c.outputs \cup c'.outputs$
11              $combinations = combinations \setminus c'$
12          if $c.outputs$ shares an element $e$ with another $c'.outputs$ then
13              $c.sharedOutputs \leftarrow c.sharedOutputs \cup e$
14          if $c.inputs$ shares an element $e$ with another $c'.inputs$ then
15              $c.sharedInputs \leftarrow c.sharedInputs \cup e$
16      return $combinations$
```

subsequences, the combinations are created following three rules (Alg.2:4-15). First, given two subsequences $t_1\text{t}t_2$ and $t_3\text{t}t_4$, if $t_1 = t_3$, then we merge both subsequences into a new combination (Alg.2:4-7). Later, given two different combinations c and c', if they share the same *output*, i.e., $c.output = c'.output$, these two combinations are merged (Alg.2:9-11). Finally, if the intersection between two combinations is not the empty set, we have to record which elements are shared by both combinations (Alg.2:12-15).

After creating all the possible combinations, for each combination c (Alg.1:12), the algorithm creates a new task t' equal to the original activity t of the current model (Alg.1:13). Then, it removes from $I(t)$ all the tasks shared with *c.inputs*, but keeping the tasks that are in *c.sharedInputs* (Alg.1:14). On the other hand, for the new task t', it retains only the elements in $I(t')$ that are contained in *c.inputs* (Alg.1:15). The same process is applied for the outputs of both t and t' but with *c.outputs* and *c.sharedOutputs* (Alg.1:16-1:17). If both the inputs and outputs of these tasks are not empty (Alg.1:18), they are included in ind_0 (Alg. 1:19). Otherwise the model goes back to its previous state and tries with a new combination. If the new task is included, the model is repaired (Alg.1:20) and the unused arcs are removed (Alg.1:21). In order to evaluate the models (Alg.1:22), we based the quality of a solution on three criteria: *fitness replay*, *precision* and *simplicity*. To measure these criteria we used the hierarchical metric defined in [10]. If the new model is better, the best individual ind_{best} is replaced with ind_0 (Alg.1:26). Otherwise the model goes back to its previous state and repeats the process with a new combination.

The main drawback of the heuristic followed to detect the possible duplicate tasks of the log (Alg.1:3 [5]) is that it does not cover all the search space, particularly with tasks involved in a length-two-loop situation, as it breaks the rule of two tasks sharing the same input and output dependencies. To solve this,

Table 1: Results for the 18 logs with the initial solutions of ProDiGen and HM.

		Logs																	
		Alpha	Folded	Loop	Fig6p25	betaSimpl.	flighCar	Fig5p19	Fig5p1AND	Fig5p1OR	Fig6p10	Fig6p31	Fig6p33	Fig6p34	Fig6p38	Fig6p39	Fig6p42	Fig6p9	RelProc
ProDiGen	C	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	P	0.8	0.75	0.79	0.75	0.86	0.81	0.9	0.76	0.73	0.78	0.61	0.65	0.73	0.93	0.93	0.7	0.79	0.89
	S	0.3	0.3	0.29	0.29	0.3	0.29	0.3	0.29	0.29	0.29	0.26	0.28	0.29	0.31	0.3	0.28	0.3	0.30
ProDiGen + LS	C	1.0	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	1.0	*1.0*	*1.0*	*1.0*
	P	0.85	*1.0*	*0.99*	*0.97*	*0.94*	*1.0*	*1.0*	*1.0*	*1.0*	*0.96*	*1.0*	*1.0*	*0.93*	*1.0*	0.94	*1.0*	*1.0*	*0.95*
	S	0.31	*0.3*	*0.3*	*0.3*	*0.31*	*0.31*	*0.31*	*0.33*	*0.33*	*0.31*	*0.31*	*0.31*	*0.31*	*0.33*	0.3	*0.3*	*0.32*	*0.32*
HM	C	1.0	1.0	1.0	1.0	1.0	0.32	1.0	0.67	1.0	1.0	1.0	1.0	0.41	0.0	0.0	0.07	0.21	1.0
	P	0.72	0.75	0.79	0.73	0.86	0.81	0.9	0.75	0.67	0.78	0.56	0.6	0.76	0.57	0.6	0.64	0.95	0.89
	S	0.3	0.3	0.29	0.28	0.31	0.29	0.31	0.33	0.31	0.29	0.26	0.27	0.29	0.28	0.29	0.28	0.32	0.30
HM + LS	C	1.0	*1.0*	*1.0*	1.0	1.0	0.32	*1.0*	0.67	*1.0*	*1.0*	*1.0*	*1.0*	0.72	*1.0*	0.53	0.36	0.21	*1.0*
	P	0.81	*1.0*	*0.99*	0.94	0.93	0.82	*1.0*	1.0	*1.0*	*0.96*	*1.0*	*1.0*	0.98	*1.0*	0.94	0.95	0.95	*0.95*
	S	0.31	*0.3*	*0.3*	0.31	0.32	0.30	*0.31*	0.35	*0.33*	*0.31*	*0.31*	*0.31*	0.32	*0.33*	0.32	0.31	0.32	*0.32*

we have to make all the process iterative: when for a task t, $max(min(|t >_L t'|, |t' >_L t|), 1)$ is greater than 1, i.e, t is detected as a duplicate activity, the upper bound for all the tasks t' that directly follow t must be updated, because these tasks will now have multiple tasks with the same label as input. Hence, if a task t is correctly duplicated in the model (Alg.1:26), we add the tasks that directly follow t —and that weren't detected as possible duplicated tasks in the first step— into *potentialDuplicatesL2L* (Alg.1:27). Therefore, the last step of the algorithm (Alg.1:31) involves a new execution of the function *localSearch* (Alg.1:7) but with *potentialDuplicatesL2L* instead of *potentialDuplicates*. In this second and final execution, the subsequences are obtained from the process model —note that in the first execution the subsequences were extracted from the log. Therefore, the algorithm parses the solution, checking which one of the activities with the same label $t' \in I(t)$ were executed just before t and which activities $t'' \in O(t)$ were executed after t. Finally, it creates the combinations based on this information.

3 Experimentation

The validation of the presented approach has been done with several synthetic logs from [5,7]. We used ProDiGen [10] and HM [11] over these set of logs to retrieve the initial solutions. On the other hand, the quality of the models was measured taking into account three metrics: fitness replay (C) [8], precision (P) [1] and simplicity (S) [9] . Table 1 shows the results retrieved before applying the presented approach —the raw solutions mined with ProDiGen and HM— and after the local search. Moreover, they show information about which algorithm retrieves better results for each metric —highlighted in grey— and which solutions are equal to the original model —highlighted in *italics*.

After applying our approach over the solutions, the proposed local search was able to enhance the results in 35 out of 36 of the cases. More specifically, the algorithm was able to i) *significantly* improve the precision, and ii) to reduce the complexity of the different models by splitting the behavior of the overly connected nodes. Furthermore, our approach was able to retrieve the original model in 25 out of 36 cases.

4 Conclusions

We have presented an approach to tackle duplicate tasks in an already discovered model. Our proposal takes as starting point a model without duplicate tasks and its respective log, and based on the local information of the log and the causal dependencies of the input mined model, it improves the comprehensibility of the solution. The presented approach has been validated with 36 different models with duplicate tasks. Results conclude that this local search is able to detect all the potential duplicate tasks in the log, and enhance the comprehensibility of the final model, by improving its fitness replay, precision and simplicity.

Acknowledgments

This work was supported by the Spanish Ministry of Economy and Competitiveness under project TIN2014-56633-C3-1-R, and the Galician Ministry of Education under the projects EM2014/012 and CN2012/151.

References

1. Adriansyah, A., Munoz-Gama, J., Carmona, J., van Dongen, B.F., van der Aalst, W.M.P.: Alignment based precision checking. In: BPM. (2012) 137–149
2. Broucke, S.K.V.: Advances in Process Mining. PhD thesis
3. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity. International Journal of Cooperative Information Systems **23**(1) (2014)
4. Carmona, J., Cortadella, J., Kishinevsky, M.: A region-based algorithm for discovering petri nets from event logs. In: BPM. Springer (2008) 358–373
5. de Medeiros, A.: Genetic Process Mining. PhD thesis, TU/e (2006)
6. Goedertier, S., Martens, D., Vanthienen, J., Baesens, B.: Robust process discovery with artificial negative events. The Journal of Machine Learning Research **10** (2009) 1305–1340
7. Li, J., Liu, D., Yang, B.: Process mining: Extending α-algorithm to mine duplicate tasks in process logs. In: Advances in Web and Network Technologies, and Information Management. (2007) 396–407
8. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Information Systems **33**(1) (2008) 64–95
9. Sánchez-González, L., Garca, F., Mendling, J., Ruiz, F., M.Piattini: Prediction of business process model quality based on structural metrics. In: Conceptual Modeling ER 2010. Volume 6412. (2010) 458–463
10. Vázquez-Barreiros, B., Mucientes, M., Lama, M.: ProDiGen: Mining complete, precise and minimal structure process models with a genetic algorithm. Information Sciences **294** (2015) 315–333
11. Weijters, A., van der Aalst, W.M.P., de Medeiros, A.: Process mining with the heuristics miner-algorithm. Technische Universiteit Eindhoven **166** (2006)

A Method for Assessing Parameter Impact on Control-Flow Discovery Algorithms

Joel Ribeiro[1] and Josep Carmona[1]

Universitat Politècnica de Catalunya, Spain.
{jribeiro, jcarmona}@cs.upc.edu

Abstract. Given an event log L, a control-flow discovery algorithm f, and a quality metric m, this paper faces the following problem: what are the parameters in f that mostly influence its application in terms of m when applied to L? This paper proposes a method to solve this problem, based on *sensitivity analysis*, a theory which has been successfully applied in other areas. Clearly, a satisfactory solution to this problem will be crucial to bridge the gap between process discovery algorithms and final users. Additionally, recommendation techniques and meta-techniques like determining the *representational bias* of an algorithm may benefit from solutions to the problem considered in this paper. The method has been evaluated over a set of logs and the flexible heuristic miner, and the preliminary results witness the applicability of the general framework described in this paper.

1 Introduction

Control-flow discovery is considered as one of the crucial features of Process Mining [13]. Intuitively, discovering the control-flow of a process requires to analyze its executions and extract the causality relations between activities which, taken together, illustrate the structure and ordering of the process under consideration.

There are many factors that may hamper the applicability of a control-flow discovery algorithm. On the one hand, the log characteristics may induce the use of particular algorithms, e.g., in the presence of *noise* in the log it may be advisable to consider a noise-aware algorithm. On the other hand, the *representational bias* of an algorithm may hinder its applicability for elicitating the process underlying in a log.

Even in the ideal case where the more suitable control-flow discovery algorithm is used for tackling the discovery task, it may be the case that the default algorithm's parameters (designed to perform well over different scenarios) are not appropriate for the log at hand. In that case, the user is left alone in the task of configuring the best parameter values, a task which requires a knowledge of both the algorithm and the log at hand.

In this paper we present a method to automatically assess the impact of parameters of control-flow discovery algorithms. In our approach, we use an efficient technique from the discipline of sensitivity analysis for exploring the parameter search space. In the next section, we charaterize this sensitivity analysis technique and relate it with other work in the literature for similar purposes done in other areas.

We consider three direct applications of the method presented in this paper:

(A) As an aid to users of control-flow discovery algorithms: given a log, an algorithm and a particular quality metric the user is interested in, a method like the one presented in this paper will indicate the parameters to consider. Then the user will be able to influence (by assigning meaningful values to these parameters) the discovery experiment.

(B) As an aid for recommending control-flow discovery algorithms: current recommendation systems for control-flow process discovery (e.g., [9]) do not consider the parameters of the algorithms. Using the methodology of this paper, one may determine classes of parameters whose impact refer to the same quality metric, and those can be offered as modes of the same algorithm tailored to specific metrics. Hence, the recommendation task (i.e., the selection of a discovery algorithm) may then be guided towards a better use of a control-flow technique.

(C) As a new form of assessing the representational bias of an algorithm: given a log and an algorithm, it may well be the case that the impact of most of the algorithm's parameters is negligible. In that case, then if additionally the result obtained is not satisfactory, one may conclude that this is not the right algorithm for the log at hand.

The rest of the paper is organized as follows: Section 2 illustrates the contribution and provides related work. Section 3 provides the necessary background and main definitions. Then, Section 4 presents the main methodology of this paper, while Section 5 provides a general discussion on its complexity. Finally, Section 6 concludes the paper.

2 Related Work and Contribution

The selection of parameters for executing control-flow algorithms is usually a challenging issue. The uncertainty of the inputs, the lack of information about parameters, the diversity of outputs (i.e., the different process model types), and the difficulty of choosing a comprehensive quality measurement for assessing the output of a control-flow algorithm make the selection of parameters a difficult task.

The *parameter optimization* is one of the most effective approaches for parameter selection. In this approach, the parameter space is searched in order to find the best parameters setting with respect to a specific quality measure. Besides the aforementioned challenges, the main challenge of this approach is to select a robust strategy to search the parameter space. Grid (or exhaustive) search, random search [2], gradient descent based search [1] and evolutionary computation [7] are typical strategies, which have proven to be effective in optimization problems, but they are usually computationally costly. [16,6,3] are examples of parameter optimization applications on a control-flow algorithm. Besides the fact that only a single control-flow algorithm is considered, all of these approaches rely on quality measurements that are especially designed to work on a specific type of process model.

A different approach, which may also be used to facilitate the parameter optimization, is known as *sensibility analysis* [11] and consists of assessing the influence of the inputs of a mathematical model (or system) on the model's output. This information may help on understanding the relationship between the inputs and the output of the model, or identifying redundant inputs in specific contexts. Sensibility methods range from variance-based methods to screening techniques [11]. One of the advantages of screening is that it requires a relatively low number of evaluations when compared to other approaches. The *Elementary Effect* (EE) method [8,4,5] is a screening technique for sensibility analysis that can be applied to identify non-influential parameters of computationally costly algorithms. In this paper, the EE method is applied to assess the impact of the parameters of control-flow algorithms.

3 Preliminaries

This section contains the main definitions used in this paper.

3.1 Event Log and Process Model

Process data describe the execution of the different process events of a business process over time. An *event log* organizes process data as a set of process instances, where a process instance represents a sequence of events describing the execution of activities (or tasks).

Definition 1 (Event Log). *Let T be a set of events, T^* the set of all sequences (i.e., process instances) that are composed of zero or more events of T, and $\delta \in T^*$ a process instance. An event log L is a set of process instances, i.e., $L \in \mathcal{P}(T^*)$.*[1]

A *process model* is an activity-centric model that describes the business process in terms of activities and their dependency relations. Petri nets, Causal nets, BPMN, and EPCs are examples of notations for modeling these models. For an overview of process notations see [13]. A process model can be seen as an abstraction of how work is done in a specific business. A process model can be discovered from process data by applying some control-flow algorithm.

3.2 Control-Flow Algorithm

A control-flow algorithm is a process discovery technique that can be used for translating the process behavior described in an event log into a process model. These algorithms may be driven by different discovery strategies and provide different functionalities. Also, the execution of a control-flow algorithm may be constrained (controlled) by some parameters.

[1] $\mathcal{P}(X)$ denotes the powerset of some set X.

Definition 2 (Algorithm). *Let L be an event log, P a list of parameters, and R a process model. An (control-flow) algorithm A is defined as a function $f^A : (L, P) \rightarrow R$ that represents in R the process behavior described in L, and it is constrained by P. The execution of f^A is designated as a **discovery experiment**.*

3.3 Quality Measure

A *measure* can be defined as a measurement that evaluates the quality of the result of an (control-flow) algorithm. A measure can be categorized as follows [13].

Simplicity measure: quantifies the results of an algorithm (i.e., a process model mined from a specific event log) in terms of readability and comprehension. The number of elements in the model is an example of a simplicity measure.

Fitness measure: quantifies how much behavior described in the log complies with the behavior represented in the process model. The fitness is 100% if the model can describe every trace in the log.

Precision measure: quantifies how much behavior represented in the process model is described in the log. The precision is 100% if the log contains every possible trace represented in the model.

Generalization measure: quantifies the degree of abstraction beyond observed behavior, i.e., a general model will accept not only traces in the log, but some others that generalize these.

Definition 3 (Measure). *Let R be a process model and L an event log. A measure M is defined by*

- *a function $g^M : (R) \rightarrow \mathbb{R}$ that quantifies the quality of R, or*
- *a function $g^M : (R, L) \rightarrow \mathbb{R}$ that quantifies the quality of R according to L.*

*The execution of g^M is designated as a **conformance experiment**.*

3.4 Problem Definition

Given an event log L, a control-flow algorithm A constrained by the list of parameters $P = [p_1 = v_1, ..., p_k = v_k]$, and a quality measure M: *Assess the impact of each parameter $p \in P$ on the result of the execution of A over L, according to M.*

4 The Elementary Effect Method

The *Elementary Effect* (EE) method [8,4,5] is a technique for sensibility analysis that can be applied to identify non-influential parameters of control-flow algorithms, which usually are computationally costly for estimating other sensitivity analysis measures (e.g., variance-based measures). Rather than quantifying the exact importance of parameters, the EE method provides insight into the contribution of parameters to the results quality.

One of the most efficient EE methods is based on Sobol quasi-random numbers [12] and a radial OAT strategy [5].[2] The main idea is to analyze the parameter space by performing experiments and assessing the impact of changing parameters with respect to the results quality. A Sobol quasi-random generator is used to determine a uniformly distributed set of points in the parameter space. Radial OAT experiments [5] are executed over the generated points to measure the impact of the parameters. This information can be used either (i) to guide on the parameters setup by prioritizing the parameters to be tuned, or (ii) as a first step towards parameter optimization.

4.1 Radial OAT Experiments

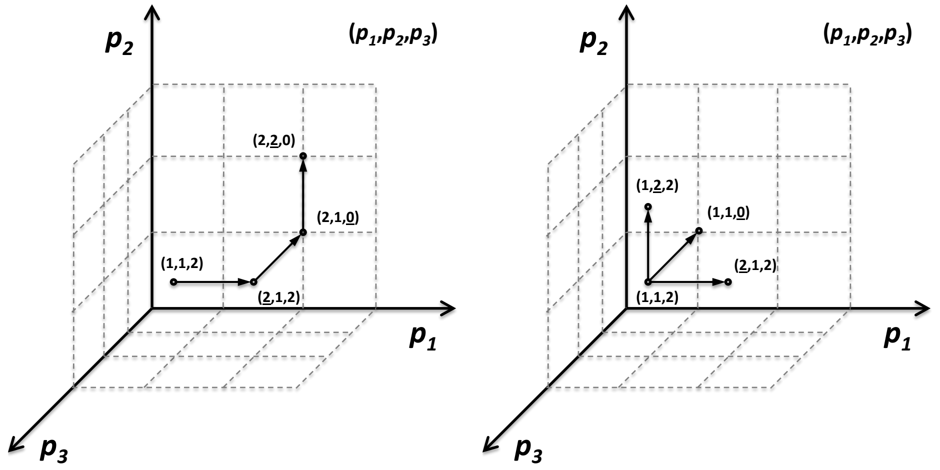
In this paper, an OAT experiment consists of a benchmark of some control-flow algorithm where the algorithm's parameters are assessed one at a time according to some quality measure. This means that $k + 1$ discovery and conformance experiments are conducted, the first to set a reference and the last k to compare the impact of changing one of the k algorithm's parameters. The parameter settings for establishing the reference and changing the parameter's values are defined by a pair of points from the parameter space. OAT experiments can use different strategies to explore these points. Figure 1 presents the most common strategies for performing OAT experiments. In the trajectory design, the parameter change compares to the point of the previous experiment. In the radial design, the parameter change compares always to the initial point. From these two, the radial design has been proven to outperform the trajectory one [10].

Radial OAT experiments can be defined as follows. First, a pair of points (α, β) is selected in the parameter space. Point α, the base point (point $(1, 1, 2)$ in Figure 1), is used as the reference parameter setting of the experiment. A discovery and conformance experiment is executed with this parameters setting to set the reference quality value. Point β, the auxiliary point (point $(2, 2, 0)$ in Figure 1), is used to compare the impact of changing the parameters, one at a time, from α to β. For each parameter $p_i \in P$, a discovery and conformance experiment is executed using the parameter values defined by α for a parameter $p_j \in P \wedge p_j \neq p_i$ and the parameter value defined by β for p_i (see the example in Figure 1b). Insight into the impact of each parameter is provided by aggregating the results of the radial OAT experiments.

Let A be a control-flow algorithm, M a given measure, and L an event log. The function $f^{A \cdot M}(L, P)$ computes the quality of the result of A over L with respect to M, where $P = [p_1 = v_1, ..., p_k = v_k]$ is the list of parameters of A.

$$f^{A \cdot M}(L, P) = \begin{cases} g^M(f^A(L, P)) & \text{if } M \text{ does not depend on a log} \\ g^M(f^A(L, P), L) & \text{otherwise} \end{cases} \tag{1}$$

[2] OAT stands for One (factor) At a Time.



(a) Trajectory.

(b) Radial.

Fig. 1: Comparison between radial and trajectory samplings for OAT experiments over 3 parameters, using the points $(1, 1, 2)$ and $(2, 2, 0)$. The underlined values identify the parameter being assessed

The elementary effect of a parameter $p_i \in P$ on a radial OAT experiment is defined by

$$EE_i = \frac{f^{A \cdot M}(L, \alpha) - f^{A \cdot M}(L, \alpha \hookleftarrow \alpha_i \cdot \beta_i)}{\alpha_i - \beta_i}, \tag{2}$$

where α, β are parameter settings of P (the base and auxiliary points), α_i and β_i are the i^{th} elements of α and β, and $f^{A \cdot M}(L, \alpha \hookleftarrow \alpha_i \cdot \beta_i)$ is the function $f^{A \cdot M}(L, \alpha')$ where α' is α with β_i replacing α_i. The measure $\mu^\star$ for p_i is defined by

$$\mu_i^\star = \frac{\sum_{j=1}^{r} |EE_i|}{r}, \tag{3}$$

where r is the number of radial OAT experiments to be executed, typically between 10 and 50 [4]. The total number of discovery and conformance experiments is $r(k + 1)$, where k is the number of parameters of A.

The impact of a parameter $p_i \in P$ is given as the relative value of $\mu_i^\star$ compared to that for the other parameters of P. A parameter $p_j \in P$ $(j \neq i)$ is considered to have more impact on the results quality than p_i if $\mu_j^\star > \mu_i^\star$. The parameters p_j and p_i are considered to have equal impact on the results quality if $\mu_j^\star = \mu_i^\star$. The parameter p_i is considered to have no impact on the results quality if $\mu_i^\star = 0$. This measure is sufficient to provide a reliable ranking of the parameters [4,5].

4.2 Sobol Numbers

Sobol quasi-random numbers (or sequences) are low-discrepancy sequences that can be used to distribute uniformly a set of points over a multidimensional space. These sequences are defined by n points with m dimensions. Table 1 presents an example of a Sobol sequence containing ten points with ten dimensions.

	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}
x_1	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000
x_2	0.7500	0.2500	0.2500	0.2500	0.7500	0.7500	0.2500	0.7500	0.7500	0.7500
x_3	0.2500	0.7500	0.7500	0.7500	0.2500	0.2500	0.7500	0.2500	0.2500	0.2500
x_4	0.3750	0.3750	0.6250	0.8750	0.3750	0.1250	0.3750	0.8750	0.8750	0.6250
x_5	0.8750	0.8750	0.1250	0.3750	0.8750	0.6250	0.8750	0.3750	0.3750	0.1250
x_6	0.6250	0.1250	0.8750	0.6250	0.6250	0.8750	0.1250	0.1250	0.1250	0.3750
x_7	0.1250	0.6250	0.3750	0.1250	0.1250	0.3750	0.6250	0.6250	0.6250	0.8750
x_8	0.1875	0.3125	0.9375	0.4375	0.5625	0.3125	0.4375	0.9375	0.9375	0.3125
x_9	0.6875	0.8125	0.4375	0.9375	0.0625	0.8125	0.9375	0.4375	0.4375	0.8125
x_{10}	0.9375	0.0625	0.6875	0.1875	0.3125	0.5625	0.1875	0.1875	0.1875	0.5625

Table 1: The first ten points of a ten-dimensional Sobol quasi-random sequence.

Each element of a point of a Sobol sequence consists of a numerical value between zero and one (e.g., the element representing the second dimension (d_2) of point x_5 is 0.8750). A collection of these values (the entire point or part of it) may be used to identify a specific point in a parameter space. An element of a point of a Sobol sequence can be converted into a parameter value by some normalization process. For instance, a possible normalization process for an element $e \in [0, 1]$ to one of the n distinct values of some discrete parameter p can be defined by $\lfloor e \times n \rfloor$, which identifies the index of the parameter value in p corresponding to e. Notice that the parameter space must be uniformly mapped by the normalization process (e.g., each value of a Boolean parameter must be represented by 50% of all possible elements).

Using the approach proposed in [5], a matrix of quasi-random Sobol numbers of dimensions $(r + 4{,}2k)$ can be used to analyze the elementary effects of the k parameters of a control-flow algorithm by executing r radial OAT experiments. The first k dimensions of the matrix's points define the base points, while the last k dimensions define the auxiliary points. Given that the first points of a Sobol sequence have the tendency to provide similar base and auxiliary points, it is identified in [5] the need of discarding the first four points of the sequence for the auxiliary points (i.e., the k rightmost columns should be shifted upward). Therefore, the base and auxiliary points can be computed from a Sobol sequence as follows. Let e_i^j be the element corresponding to the j^{th} dimension (d_j) of the i^{th} point (x_i) of the sequence. The i^{th} base (α^i) and auxiliary (β^i) points are defined as following.

$$\alpha^i = (e_i^1, e_i^2, ..., e_i^j) \; and \; \beta^i = (e_{i+4}^{j+1}, e_{i+4}^{j+2}, ..., e_{i+4}^{2j}). \qquad (4)$$

4.3 Example: The FHM

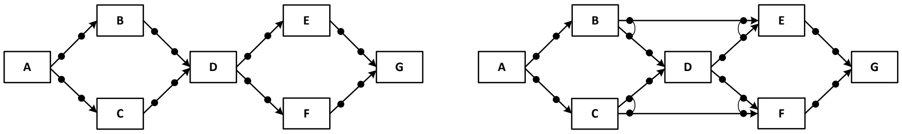
The following example is used to illustrate the analysis of the parameter space of an algorithm in order to assess the impact of the algorithm's parameters on the results quality. Let us consider an event log that is characterized by two distinct traces: $ABDEG$ and $ACDFG$. The frequency of any of these traces is high enough to not be considered as noise. The behavior described by these traces does not contain any kind of loop or parallelism, but it does contain two long-distance dependencies: $B \Rightarrow E$ and $C \Rightarrow F$. Let us also consider the Flexible Heuristics Miner (FHM) [17] as the control-flow algorithm to explore the parameter space in order to assess the impact of the FHM's parameters on the results quality. The parameters of the FHM are summarized in Table 2. Notice that every parameter of the FHM is continuous, with a range between zero and one. The *relative-to-best* and the *long-distance* thresholds are optional. The former is only considered with the *all-tasks-connected* heuristic. The latter is only taken into account when the *long-distance dependencies* option is activated.

Parameter	*Domain*	*Optional?*
Relative-to-best Threshold	$[0, 1]$	Yes
Dependency Threshold	$[0, 1]$	No
Length-one-loops Threshold	$[0, 1]$	No
Length-two-loops Threshold	$[0, 1]$	No
Long-distance Threshold	$[0, 1]$	Yes

Table 2: The parameters of the Flexible Heuristics Miner [17].

Figure 2 presents the two possible process models that can be mined with the FHM on the aforementioned event log, using all combinations of parameter values. Figure 2a shows the resulting Causal net where long-distance dependencies are not taken into account. Figure 2b shows the resulting Causal net with the long-distance dependencies. Notice that, depending on the quality measure, the quality of these process models may differ (e.g., the precision of the model with long-distance dependencies is higher than the other one). One may be interested on the exploration of the FHM's parameter space to get the process model that fulfills best some quality requirements.

The analysis of the parameter space of the FHM starts with the generation of the Sobol numbers. Let us consider that, for this analysis, one wants to execute $r = 30$ radial OAT experiments for assessing the elementary effects of the $k = 5$ FHM's parameters. So, a matrix of Sobol numbers of dimensions $(30 + 4{,}2 \times 5)$ has to be generated (cf. Section 4.2). Table 1 shows the first ten points of this matrix. Table 3 presents the first five base and auxiliary points as well as the parameter values corresponding to these points. Notice that the parameters are represented in the points according to the same ordering in Table 2 (i.e., the first element of a point represents the first parameter and so on). The normalization



(a) The Causal net without long-distance dependency relations.

(b) The Causal net with long-distance dependency relations.

Fig. 2: The process models that can be mined with the FHM.

process in this example is defined as follows. For the non-optional parameters (cf. Table 2), an element $e \in [0,1]$ of a point of a Sobol sequence can be directly used to represent the value of the parameter. For the optional parameters, an element $e \in [0,1]$ of a point of a Sobol sequence is normalized to a value $e' \in [0,2]$, which maps the parameter space uniformly (i.e., the value of the parameter and whether or not the parameter is enabled). If $e' \leq 1$ then e' is assigned as the value of the parameter; the parameter is disabled otherwise.

Point	*Base*	*Auxiliary*
1	(.5000, .5000, .5000, .5000, .5000)	(.6250, .8750, .3750, .3750, .1250)
2	(.7500, .2500, .2500, .2500, .7500)	(.8750, .1250, .1250, .1250, .3750)
3	(.2500, .7500, .7500, .7500, .2500)	(.3750, .6250, .6250, .6250, .8750)
4	(.3750, .3750, .6250, .8750, .3750)	(.3125, .4375, .9375, .9375, .3125)
5	(.8750, .8750, .1250, .3750, .8750)	(.8125, .9375, .4375, .4375, .8125)
...	...	...

(a) The first five base and auxiliary points.

Point	*Base*	*Auxiliary*
1	(–, 0.50, 0.50, 0.50, –)	(–, 0.88, 0.38, 0.38, 0.25)
2	(–, 0.25, 0.25, 0.25, –)	(–, 0.13, 0.13, 0.13, 0.75)
3	(0.50, 0.75, 0.75, 0.75, 0.50)	(0.75, 0.63, 0.63, 0.63, –)
4	(0.75, 0.38, 0.63, 0.88, 0.75)	(0.63, 0.44, 0.94, 0.94, 0.63)
5	(–, 0.88, 0.13, 0.38, –)	(–, 0.94, 0.44, 0.44, –)
...	...	...

(b) The parameter values for the first five base and auxiliary points. The wildcard value '–' identifies that the parameter is disabled.

Table 3: The first five points of the Sobol numbers.

Table 4 presents the radial sampling for the first radial OAT experiment (first point in Table 3) as well as the result of the execution of $f^{A \cdot M}(L, P)$ and the elementary effect EE for each parameter. For executing $f^{A \cdot M}(L, P)$, A is the

FHM, M the *Node Arc Degree* measure[3], and L the aforementioned event log. The elementary effects are computed as described in Section 4.1.[4] Notice that the elementary effect of a parameter can only be computed when the base and auxiliary points provide distinct parameter values (e.g., in Table 4, the first parameter is not assessed because it is disabled in both base and auxiliary points).

Parameter Values	*Result*	*Elementary Effect*
P	$f^{A \cdot M}(L, P)$	EE_i
(–, 0.50, 0.50, 0.50, –)	2.154	
($\underline{–}$, 0.50, 0.50, 0.50, –)		
(–, $\underline{0.88}$, 0.50, 0.50, –)	2.154	0.0
(–, 0.50, $\underline{0.38}$, 0.50, –)	2.154	0.0
(–, 0.50, 0.50, $\underline{0.38}$, –)	2.154	0.0
(–, 0.50, 0.50, 0.50, $\underline{0.25}$)	2.316	0.162

Table 4: Radial sampling for the first radial OAT experiment. The first line corresponds to the base point, while the others consist of the base point in which the element regarding a specific parameter is replaced by that from the auxiliary point; the underlined values identify the replaced element and the parameter being assessed.

Table 5 presents the results of the analysis of the FHM's parameter space. The results identify the long-distance threshold as the only parameter to take into account for the parameter exploration. As expected, all other parameters have no impact on the results quality. This is explained by the fact that the log does not contain any kind of loop or noise. Notice that the $\mu^\star$ absolute value does not provide any insight into how much a parameter influences the results quality. Instead, the $\mu^\star$ measurement provides insight into the impact of a parameter on the results quality, compared to others.

Parameter	$\mu^\star$
Dependency Threshold	0.0
Relative-to-best Threshold	0.0
Length-one-loops Threshold	0.0
Length-two-loops Threshold	0.0
Long-distance Threshold	0.113

Table 5: The $\mu^\star$ values of the FHM's parameters.

[3] The *Node Arc Degree* measure consists of the average of incoming and outgoing arcs of every node of the process model.

[4] For computing EE_i, $\alpha_i - \beta_i$ is considered to be 1 when the parameter is changed from a disabled to an enabled state, or the other way around (e.g., the last parameter in Table 4).

5 Application

The EE method presented in the previous section can be applied to any control-flow algorithm constrained by many parameters, using some event log and a measure capable of quantifying the quality of the result of the algorithm. The presented method can be easily implemented on some framework capable of executing discovery and conformance experiments (e.g., ProM [15] or CoBeFra [14]). Several open-source generators of Sobol numbers are available on the web.

The computational cost of our approach can be defined as follows. Let L be an event log, A a control-flow algorithm constrained by the list of parameters $P = [p_1 = v_1, ..., p_k = v_k]$, and M a quality measure. The computational cost of a discovery experiment using A (with some parameter setting) over L is given by C_D. Considering R as the result of a discovery experiment, the computational cost of a conformance experiment over R and L (or just R) with regard to M is given by C_C. Therefore, the computational cost of a radial OAT experiment is given by $C_E = (k + 1)(C_D + C_C)$, where k is the number of parameters of A. The computational cost of the EE method based on r radial OAT experiments is given by $C = r(k + 1)(C_D + C_C)$.

5.1 Perfomance Optimization

Considering that both discovery and conformance experiments may be computationally costly, performance may become a critical issue for the application of this method. This issue can be partially addressed by identifying a set of potentially irrelevant parameters, and considering those parameters as a group. Then, by adjusting the $\mu^\star$ measurement to work with groups of two or more parameters [4], the group of parameters can be analyzed together using radial experiments that iterate over all elements of the same group simultaneously.

Suppose, for instance, that it is known that a given log does not have loops. So, for the FHM's parameters, the *length-one-loops* and *length-two-loops* thresholds may be grouped in order to avoid the execution of discovery and conformance experiments that are not relevant for the analysis. Recalling the example presented in Section 4.3, the radial experiments will iterate over one group of two parameters and three indepedent parameters (i.e., the *dependency*, the *relative-to-best*, and the *long-distance* thresholds). This means that, for the group of parameters, all elements of the same group are replaced simultaneously by the corresponding elements from the auxiliary point. Table 6 presents the adjusted radial sampling presented in Table 4. The first line corresponds to the base point, while the others consist of the base point in which the element(s) regarding a specific parameter (or group of parameters) is replaced by that from the auxiliary point; the underlined values identify the replaced element(s) and the parameter (or group of parameters) being assessed.

Parameter Values
(–, 0.50, 0.50, 0.50, –)
($\underline{–}$, 0.50, 0.50, 0.50, –)
(–, $\underline{0.88}$, 0.50, 0.50, –)
(–, 0.50, $\underline{0.38}$, $\underline{0.38}$, –)
(–, 0.50, 0.50, 0.50, $\underline{0.25}$)

Table 6: Radial sampling for the first radial experiment considering a group of parameters.

The elementary effect of a group of parameters $G \subseteq P$ on a radial experiment is defined by

$$EE_G = \frac{f^{A \cdot M}(L, \alpha) - f^{A \cdot M}(L, \alpha \hookleftarrow \alpha_G \cdot \beta_G)}{dist(\alpha_G, \beta_G)}, \tag{5}$$

where α, β are parameter settings of P (the base and auxiliary points), α_G and β_G are the elements of G in α and β, and $f^{A \cdot M}(L, \alpha \hookleftarrow \alpha_G \cdot \beta_G)$ is the function $f^{A \cdot M}(L, \alpha')$ where α' is α with β_G replacing α_G. The function $dist(A, B)$ computes the distance between A and B (e.g., the Euclidean distance). The measure $\mu^{\star}$ for G is defined by

$$\mu^{\star}_G = \frac{\sum_{j=1}^{r} |EE_G|}{r}, \tag{6}$$

where r is the number of radial experiments to be executed. The total number of discovery and conformance experiments depends on the number of groups and independent parameters being assessed.

6 Conclusions and Future Work

To the best of our knowledge, this work is the first in presenting a methodology to assess the impact of parameters in control-flow discovery algorithms. The method relies on a modern sensitivity analysis technique that requires considerably less exploration than traditional ones such as genetic algorithms or variance-based methods.

In this work, we have applied the methodology on the Flexible Heuristics Miner algorithm using 13 event logs. The results suggest the effectiveness of the method. We have noticed that simple conformance measures (and, thus, less computationally costly) are as good as any other complex measure for assessing the parameters influence. Nevertheless, we acknowledge that more experiments are necessary to get a better insight.

Future work is mainly oriented towards addressing three aspects, which are mainly addressed to apply the method of this paper to other control-flow algorithms. First, we are interested in the algorithmic perspective in order to study the most efficient form of assessing the impact of a parameter, with the method presented in this paper as a baseline. Second, we will try to incorporate the methodology described in this paper in the RS4PD, a recommender system for process discovery [9]. Finally, the application of the presented method with other goals, e.g., estimating the representational bias of control-flow discovery algorithms may be explored.

Acknowledgments. This work as been partially supported by funds from the Spanish Ministry for Economy and Competitiveness (MINECO) and the European Union (FEDER funds) under grant COMMAS (ref. TIN2013-46181-C2-1-R).

References

1. Y. Bengio. Gradient-Based Optimization of Hyperparameters. *Neural computation*, 12(8):1889–1900, 2000.
2. J. Bergstra and Y. Bengio. Random Search for Hyper-Parameter Optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.
3. A. Burattin and A. Sperduti. Automatic Determination of Parameters' Values for Heuristics Miner++. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8, July 2010.
4. F. Campolongo, J. Cariboni, and A. Saltelli. An Effective Screening Design for Sensitivity Analysis of Large Models. *Environmental Modelling & Software*, 22(10):1509 – 1518, 2007.
5. F. Campolongo, A. Saltelli, and J. Cariboni. From Screening to Quantitative Sensitivity Analysis. A Unified Approach. *Computer Physics Communications*, 182(4):978–988, 2011.
6. L. Ma. How to Evaluate the Performance of Process Discovery Algorithms: A Benchmark Experiment to Assess the Performance of Flexible Heuristics Miner. Master's thesis, Eindhoven University of Technology, Eindhoven, 2012.
7. Z. Michalewicz and M. Schoenauer. Evolutionary Algorithms for Constrained Parameter Optimization Problems. *Evolutionary Computation*, 4(1):1–32, March 1996.
8. M.D. Morris. Factorial Sampling Plans for Preliminary Computational Experiments. *Technometrics*, 33(2):161–174, April 1991.
9. J. Ribeiro, J. Carmona, M. Misir, and M. Sebag. A Recommender System for Process Discovery. In S. Sadiq, P. Soffer, and H. Vlzer, editors, *Business Process Management*, volume 8659 of *Lecture Notes in Computer Science*, pages 67–83. Springer International Publishing, 2014.
10. A. Saltelli, P. Annoni, I. Azzini, F. Campolongo, M. Ratto, and S. Tarantola. Variance Based Sensitivity Analysis of Model Output. Design and Estimator for the Total Sensitivity Index. *Computer Physics Communications*, 181(2):259 – 270, 2010.
11. A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, and S. Tarantola. *Global Sensitivity Analysis: The Primer.* Wiley, 2008.
12. I.M. Sobol. Uniformly Distributed Sequences With an Additional Uniform Property. *USSR Computational Mathematics and Mathematical Physics*, 16(5):236 – 242, 1976.
13. W.M.P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes.* Springer, Berlin, 2011.

14. S. vanden Broucke, J.D. Weerdt, B. Baesens, and J. Vanthienen. A Comprehensive Benchmarking Framework (CoBeFra) for conformance analysis between procedural process models and event logs in ProM. In *IEEE Symposium on Computational Intelligence and Data Mining*, Grand Copthorne Hotel, Singapore, 2013. IEEE.
15. H.M.W. Verbeek, J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. ProM 6: The Process Mining Toolkit. In *Demo at the 8th International Conference on Business Process Management*, volume 615 of *CEUR-WS*, pages 34–39. 2010.
16. A.J.M.M. Weijters. An Optimization Framework for Process Discovery Algorithms. In *Proceedings of the International Conference on Data Mining, Las Vegas, Nevada, USA*, 2011.
17. A.J.M.M. Weijters and J.T.S. Ribeiro. Flexible Heuristics Miner (FHM). In *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2011, Paris, France*. IEEE, 2011.

Knowledge Driven Behavioural Analysis in Process Intelligence

Antonia Azzini, Paolo Ceravolo, Ernesto Damiani, and Francesco Zavatarelli

Computer Science Department, Università degli Studi di Milano
via Bramante, 65 - 26013 - Crema, Italy
email{*name*}.{*surname*}@unimi.it

Abstract. In this paper we illustrate how the knowledge driven Behaviour Analysis, which has been used in the KITE.it process management framework, can support the evolution of analytics from descriptive to predictive. We describe how the methodology uses an iterative three-step process: first the descriptive knowledge is collected, querying the knowledge base, then the prescriptive and predictive knowledge phases allow us to evaluate business rules and objectives, extract unexpected business patterns, and screen exceptions. The procedure is iterative since this novel knowledge drives the definition of new descriptive analytics that can be combined with business rules and objectives to increase our level of knowledge on the combination between process behaviour and contextual information.

1 Introduction

Process Intelligence (PI), i.e. the convergence of operational business intelligence [1] and real-time application integration, has gain a lot of attention in the last years, especially around applications involving sensor networks [2]. The final aim is to provide more accurate and fast decisions on the strategic and operational management levels. Most of the current studies on PI focus on the analysis of the process behavior and support performance improvement limited to this aspects [3]. But, descriptive analysis is contextual in nature [4], its value is clarified by the knowledge you have on a process, for instance in terms of business rules that apply and constrain a process [5]. In particular our claim is that, to insert PI into a consistent knowledge acquisition process [6], the level of its maturity and practical implementation has to evolve in the following directions:

- Not limit their analysis to process behavior but enlarge the scope to any other auxiliary data that is connected to process execution.
- Not limit to descriptive analysis but exploit the acquired knowledge for predictive analysis.

For this purpose, we introduced the KITE Knowledge Acquisition Process [6], a methodology dealing with the evolution of analytics from descriptive to prescriptive, to predictive intention. In KITE an initial set of metrics offer the initial *descriptive knowledge*. Then our analytics support the evaluation of process instances based on their consistency with policies, business rules and KPI, defined at the strategic level.

These constrains are refereed in general as *prescriptions*. Process instances violating prescriptions offer a crucial source of knowledge acquisition as *predictive analytics* can evaluate the incidence of specific variables on violations, to then derive predictive knowledge. Indeed, predictive analytics involves searching for meaningful relationships among variables and representing those relationships in models. There are response variables - things we are trying to predict, in our case violations to prescriptions. There are explanatory variables or predictors - things we observe. To generalise, as much as possible our predictive power, predictors in our case are any data related to resource auxiliary to process execution. Actually, in our approach, metrics measure process behaviour in an extended sense, as the information retrieved is not limited to the workflow, but include data related to any resource auxiliary to the process execution, as already discussed in [7]. Our approach differs from traditional predictive analytics because it is centred on the knowledge provided by the organization via Business Rules and other documentation. This approach was framed by KITE in the firm belief that it can put in contact PI and predictive analytics with Knowledge Management.

The paper is organized as follow. Section 2 starts the discussion with the related work. Sections 3 and 4 describe the KITE framework. Section 5 describes how KITE knowledge acquisition process works. Section 6 deals with behavioural and predictive analysis. Section 7 illustrate our ideas through an example. Section 8 proposes some conclusions.

2 Related Work

Predictive analytics applied to process monitoring is often limited, or strongly depended, to temporal analysis. For instance in [8] temporal logic rules are adopted to define business constraints. The approach is then focused on the evaluation of these constraints at execution time, to generate alerts that can prevent the development of violations. In [9], the authors present a set of approaches based on annotated transition systems containing time information extracted from event logs. The aim is again to check time conformance at execution time, as executions not aligned with annotated transitions predict the remaining processing time, and recommend countermeasures to the end users. An approach for prediction of abnormal termination of business processes has been presented in [10]. Here, a fault detection algorithm (local outlier factor) is used to estimate the probability of abnormal termination. Alarms are provided to early notify probable abnormal terminations to prevent risks rather than merely reactive correction of risk eventualities. Other approaches go beyond temporal analysis extending predictive analytics to include ad-hoc contextual information. In [11], a clustering approach on SLA properties is coupled with behavioral analysis to discovered and model performance predictors. In [12], the authors propose an approach running statistical analysis on process-related data, notably the activities performed, their sequence, resource availability, capabilities and interaction patterns. In [13], the authors propose an approach for Root Cause Analysis based on classification algorithms. After enriching a log with information like workload, occurrence of delay and involvement of resources, they use decision trees to identify the causes of overtime faults. In such an analysis, the availability of attributes/features that may explain the root cause of some phenomena is crucial.

On the side of knowledge acquisition procedures the literature presents several works specifically oriented to the area of business process management [14]. However only a few are really considering analytics as a key element of this process. For instance in [15] the authors exploit the notion of knowledge maintenance process. process mining is applied to analyze the knowledge maintenance logs to discover process and then construct a more appropriate knowledge maintenance process model. The proposed approach has been applied in the knowledge management system.

Our work is characterized by the introduction of an extended notion of process behavior that provide a generalized systematic approach to captures process features beyond workflow execution. This element is the exploited within a knowledge acquisition methodology that exploits prescriptive and predictive analytics to acquire novel and unexpected knowledge.

3 The KITE Methodology

KITE.it is a project co-funded by the Italian Ministry for Economic Development, within the "Industria 2015" Program, in the area of "New technologies for Made in Italy" [16]. The exit from the great global crisis towards a new cycle of development requires to move from organizational and inter-organizational models, based on a strict definition of roles and organizational boundaries. In this context, KITE.it is aimed at developing a business and social cooperation framework that enables interoperability among enterprises and other knowledge workers, making available a variety of tools and technologies developed to connect the processes of an organization to those of suppliers or to involve customers in planning and assessing activities. In fact, the KITE.it framework should be capable of supporting procedures such as *i)* creation, contextualization and execution of metrics, *ii)* connection between metrics and strategic level, and *iii)* inception and capitalization of the results. The final goal is driving the monitoring process to derive previously unknown and potentially unexpected knowledge.

To circumscribe our discussion, in this paper we examine a single aspect of the KITE.it Framework, focusing on how it was extended to cover data integration and interoperability, as discussed in Section 4. Moreover, we are considering how these characteristics was exploited in guiding the Knowledge Acquisition Process, as discussed in Sec 5.

4 The KITE Knowledge Base

The KITE Knowledge Base (KKB) has to integrate a variety of heterogeneous data from the different sources composing the KITE.it Framework.

This requirement is faced adopting a graph-based model to structure and link data according to the Web Standards, the so-called Resource Description Framework. Generally speaking, the Resource Description Framework (RDF) [17] provides a standard for defining vocabularies, which can be adopted to generate directed labeled graphs [18], in which entities edges and value are associated with terms of the vocabulary. For this reason, RDF is an extremely generic data representation model that can be extended easily with any domain-specific information. Moreover, RDF is a monotonic declarative language, i.e. the acquisition of new data cannot invalidate the information previously acquired.

The atomic elements of a RDF graph are triples[1]. Triples are composed by three elements: resources, relations between resources and attributes of resources. These elements are modeled within the labelled oriented graph, as the atomic structure $< s, p, o >$ where s is subject, p is predicate and o is object, combined as shown in Figure 1.

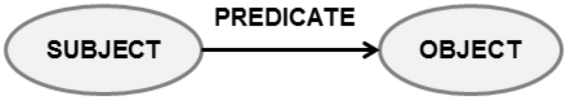


Fig. 1. RDF subject-object relation.

New information is inserted into an RDF graph by adding new triples to the data set. It is therefore easy to understand why such a representation can provide big benefits for real time business process analysis: data can be appended 'on the fly' to the existing one, and it will become part of the graph, available for any analytical application, without the need for reconfiguration or any other data preparation steps.

Assuming pairwise disjoint infinite sets I, B, L (*IRIs*[2], *Blank nodes*, *RDF Literals*).

Definition 1 *A tuple* $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$ *is called an RDF triple.*

An RDF graph G is a set of RDF triples. An interesting feature of RDF standards is that multiple graphs can be stored in a single RDF Dataset. As stated in the specifications "*An RDF Dataset comprises one graph, the default graph, which does not have a name, and zero or more named graphs, where each named graph is identified by an IRI*".

RDF standard vocabularies allow external applications to query data through SPARQL query language [19]. SPARQL is a standard query language for RDF graphs based on conjunctive queries on triple patterns, identifying paths in the RDF graph. Thus, queries can be seen as graph views. SPARQL is supported by most of the triples stores available.

If we now introduce a novel infinite set V for variables, disjoint from I, B, and L we can define SPARQL patterns as in the following.

Definition 2 *A tuple* $t \in (I \cup L \cup V) \times (I \cup V) \times (I \cup L \cup V)$ *is called a SPARQL triple pattern. Where the blank nodes act as non-distinguished variables in graph patterns.*

Definition 3 *A finite set of SPARQL triple patterns can be constructed in a Graph Pattern (GP) using OPTIONAL, UNION, FILTER and JOIN. A Basic Graph Pattern is a set of triple patterns connect by the JOIN operator.*

The semantics of SPARQL is based on the notion of mapping, defined in [20] as a partial function $\mu : V \to (I \cup L \cup B)$. Where, if GP is a graph pattern and $var(GP)$ denotes

[1] An alternative terminology adopted in documentation is *statements* or eventually *tuples*.

[2] IRIs are the RDF URI references, IRIs allow all characters beyond the US-ASCII charset.

the set of variables occurring in *GP*; given a triple pattern t and a mapping μ such that $var(t) \subseteq dom(\mu)$, μ is the triple obtained by replacing the variable in t according to μ.

In [21], the authors present a framework based on RDF for business process monitoring and analysis. They define an RDF model to represent a generic business process that can be easily extended in order to describe any specific business process by only extending the RDF vocabulary and adding new triples to the triple store. The model is used as a reference by both monitoring applications (i.e., applications producing the data to be analyzed) and analyzing tools. On one side, a process monitor creates and maintains the extension of the generic business process vocabulary either at start time, if the process is known a priori, or at runtime while capturing process execution data, if the process is not known. Process execution data is then saved as triples with respect to the extended model. On the other side, the analyzing tools may send SPARQL queries to the continuously updated process execution RDF graph.

Figure 2 shows the schema of an RDF Dataset composed by the union of two graphs. The resources describing the generic model of a business process are tagged in blue. They can represent a sequence of different tasks, each having a start/end time and having zero or more sub-tasks. The resources tagged in yellow represent domain-specific concepts describing the repair and overhaul process in avionics. In this very simple extract we defined a process, in connection with its tasks, and the customer purchasing the overhaul operations.

Once this schema is defined any process execution is stored in the KKB in terms an RDF Dataset composed of triples conforming with the schema. For instance, in 1 a legal dataset is presented.

```
av:p1 rdf:type av:Overhaul
av:p1 bpm:hasTask av:t1
av:p1 bpm:hasTask av:t2
av:t1 bpm:followedBy av:t2
av:t1 bpm:startTime "2013-06-06 10:38:45"^^xsd:date
av:t1 bpm:endTime "2013-06-06 18:12:35 "^^xsd:date
av:t1 rdf:type av:Inspect
```

(1)

5 The KITE Knowledge Acquisition Process

The methodology considers the KITE Knowledge Acquisition Process (KKAP) as an investigation over the process executions, as registered in the KKB. In particular, this methodology is organised in iterations over three fundamental steps.

- *Descriptive Knowledge*: querying triples on the process execution, or any other auxiliary resource, you have a descriptive summary of the process in terms of frequency, dimension, and central tendency [22].
- *Prescriptive Knowledge*: evaluating the achievement of the business rules or the objectives associated to a process, as well as identifying unexpected patterns, you can screen of process executions isolating exceptions that are violating some prescription [23].

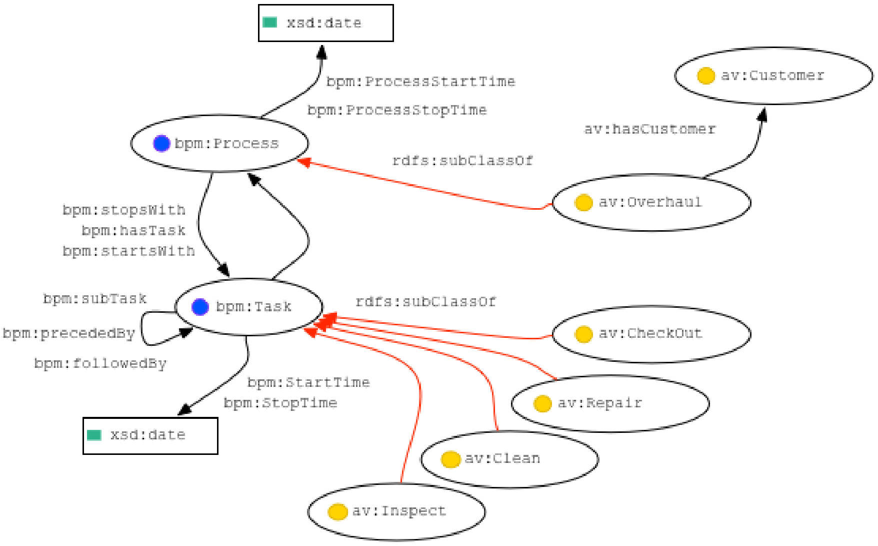


Fig. 2. RDF Representation of a generic business process.

- *Predictive Knowledge*: process executions screened by prescriptions can be further investigated evaluating the incidence of specific properties on specific partitions of the KKB. This allows to acquire novel knowledge on the process that eventually can result in new descriptive or prescriptive knowledge.

Before providing further definitions let us clarify our purpose by a simple example of two iterations.

5.1 First iteration

The engine maintenance is a very complex process performed by the aerospace industry. Generally speaking, maintenance operations are needed on a regular time basis (*Inspect Only*, *Minor Revision* or *General Revision*, according to the number of flown hours) or when a part has failed (*Out of Order*), as shown in table 1. The activities vary accordingly.

	Inspect	**Disassembly**	**Inspect**	**Repair**	**Clean**	**Assembly**	**Bench**	**Checkout**
	(I)	(DA)	**Mod.** (IM)	(R)	(C)	(A)	**Test** (BT)	(CO)
General Rev.		Yes	Yes		Yes	Yes		Yes
Minor Rev.	Yes				Yes			Yes
Inspect Only	Yes							Yes
Out of Order	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Table 1. Representation of the maintenance processes in the aerospace industry (simplified).

Suppose to focus on minor and general revision processes, and collect the duration in days of all the process executions involving the activities *Inspect*, *Clean* and *CheckOut* ($I \gg C \gg CO$ in short) in case of minor revision, or *Disassembly*, *Inspect Module*, *Clean*, *Assembly* and *CheckOut* ($DA \gg IM \gg C \gg A \gg CO$ in short) when general revision is performed. Results can be summarised as illustrated in table 2. In this way you have Descriptive Knowledge about the processes.

ProcessID	**Task Sequence**	**Duration (days)**
p12	Minor Revision	3
p31	Minor Revision	4
p33	Minor Revision	5
p39	Minor Revision	3
p11	Minor Revision	8
p05	Minor Revision	5
p101	General Revision	12
p102	General Revision	11
p103	General Revision	13
p104	General Revision	11
...	...	...

Table 2. Duration in days of process executions involving Minor and General Revision.

To acquire Prescriptive Knowledge you have to compare your data with some prescriptions. By this term here we refer to any constraint or property the business processes execution should satisfy. In the Business Process Management literature, this function is typically associated with Business Rules [24], even if their scope is not limited at assessing the business behavior but involves the business structure as well (for instance defining the corporate governance). Business Rules can derive from internal objectives and strategies or from external factors such as contractual constrains or legal requirements. However, Business Rules can also be discovered by data mining [25] or process mining [26], for instance by identifying recurrent behavior.

Once a prescription is defined you are able to partition the dataset based on the violations of this prescription. If the violation can be associated to an intensity the partitions depend on a degree, otherwise the partition is binary. For instance Business Rules could prescribe the expected duration of process executions: $Duration \leq 7\ days$ if Minor Revision and $Duration \leq 11\ days$ if General Revision. Table 3 shows the result of this operation. The prescription that have been learned from a dataset d can be applied to other datasets D, under the assumption that d is a representative sample of D.

The notion of violation is crucial in the KITE methodology as it identify an observation that is not consistent with our expectations and we would like to avoid for future executions. Investigating the incidence of specific resources on the sub set of the violations we can induce additional knowledge to support explanation or resolution of process executions violating our prescription. To draw conclusions of our example let us introduce an additional resource in our view of the dataset, as illustrated in Table 4. If we can observe a significant incidence of this resource to the subset of the violations we

ProcessID	Task Sequence	Duration	Violation
p12	Minor Revision	3	NO
p39	Minor Revision	3	NO
p31	Minor Revision	4	NO
p33	Minor Revision	5	NO
p05	Minor Revision	5	NO
p102	General Revision	11	NO
p104	General Revision	11	NO
...	...	...	...
p11	Minor Revision	8	YES
p101	General Revision	12	YES
p103	General Revision	13	YES
...	...	...	...

Table 3. Dataset partitioned according to the prescription (business rule).

can attest the acquisition of novel knowledge that should be exploited for the definition of a second iteration of the KKAP.

ProcessID	Task Sequence	Duration	Violation	Customer Type
p12	Minor Revision	3	NO	Civil
p39	Minor Revision	3	NO	Civil
p31	Minor Revision	4	NO	Civil
p33	Minor Revision	5	NO	Military
p05	Minor Revision	5	NO	Civil
p102	General Revision	11	NO	Civil
p104	General Revision	11	NO	Civil
...	...	...	...	...
p11	Minor Revision	8	YES	Military
p101	General Revision	12	YES	Military
p103	General Revision	13	YES	Military
...	...	...	...	...

Table 4. Incidence of an additional resource (customer type) on the violations.

5.2 Second iteration

As previously stated, this is an iterative process, which takes the last data as a starting point for the second iteration, see Figure 3.

We start our second iteration as shown in table 5, where some other kind of processes (*Inspect Only* and *Out of Order*) are added to the dataset for a better understanding. We also introduce additional resources, in this case the notion whether the activities were performed by internal company staff or outsourced to somebody else.

Defining another prescription we are able again to partition the dataset based on the violations of the new business rule. Consider for example to define the following

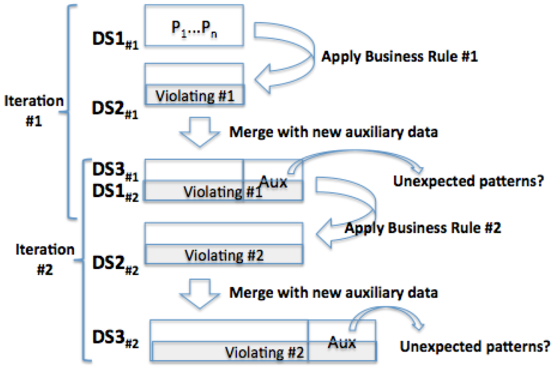


Fig. 3. iterations

prescription: operations must not be outsourced if the engine belongs to a military customer. Table 6 shows the result of this operation.

ProcessID	Task Sequence	Customer Type	Staff
p11	Minor Revision	Military	Outsourced
p101	General Revision	Military	Internal
p103	General Revision	Military	Internal
p202	Inspect Only	Military	Internal
p301	Out of Order	Military	Outsourced
...	...	...	...

Table 5. Responsibility of the activities, as performed by internal staff staff or outsourced.

ProcessID	Task Sequence	Customer Type	Staff	Violating
p101	General Revision	Military	Internal	NO
p103	General Revision	Military	Internal	NO
p202	Inspect Only	Military	Internal	NO
...	...	...	...	...
p11	Minor Revision	Military	Outsourced	YES
p301	Out of Order	Military	Outsourced	YES
...	...	...	...	...

Table 6. Violations of the second business rule.

Investigating again the incidence of a specific resource on the subset of the violation we can induce additional knowledge and support explanation. In our case, in order to draw conclusions we introduce an additional resource in our view of the dataset, as illustrated in Table 7. When we observe a significant incidence of this resource to the subset of the violations we have acquisition of novel knowledge.

ProcessID	Task Sequence	Customer Type	Staff	Violating	Certified
p101	General Revision	Military	Internal	NO	YES
p103	General Revision	Military	Internal	NO	YES
p202	Inspect Only	Military	Internal	NO	YES
...	...	...	...	...	...
p11	Minor Revision	Military	Outsourced	YES	NO
p301	Out of Order	Military	Outsourced	YES	NO
...	...	...	...	...	...

Table 7. An additional resource can lead to novel knowledge.

6 Predictive Analytics

As illustrated in [7] we extended the notion of Behavioral Analysis as a weaker form of classic behavior equivalence, where two compatible behaviors have to be equivalent with respect to activities they have in common [27]. To characterise a process execution log, for instance for detecting ordering relations among events, it is common to start by the definition of *process execution tracks*, *workflow trace* as defined in [28]. In our approach, we extended this definition by auxiliary resources, considering any data related to the events in a trace that are consistent with a graph pattern over the KKB.

As already mentioned the KKAP includes predictive analytics aimed at identifying the incidence of KKB's resources on process execution. In general, predictive analytics encompasses a variety of statistical techniques from modeling, machine learning, and data mining that uncovers relationships and patterns within large volumes of data that can be used to predict behavior and events [29]. Here we adopt the term to refer to this part of our methodology that is is forward-looking, i.e. uses past events to better understand the process. In particular our aim is to investigate data about resources auxiliary to process execution, searching for incidence with those process instances that are violating prescriptions. Now, our aim is to define how this incidence is evaluated.

The approach adopted in KITE is based on Bayesian statistics [30]. Bayesian statistics offers the theoretical framework for combining experimental and extra-experimental knowledge. In particular, Bayesian procedures, for evaluating the predictive power of a parameter in a statistical model, take into account both experimental data and information on the parameter incorporated in the so-called prior distribution[3]. This is an important point of distinction with frequentist approaches, most commonly used. The most practical consequence is that frequentist approaches impose assumptions on the distribution for both the random sample and the model tested. Different hypothesis tests have different model assumptions. For many tests, the model assumptions consist of several conditions. If any one of these conditions is not true, we do not know that the test is valid. But these assumptions cannot be easily verified on any kind of data sets, in particular when dealing with data flows acquired or consumed at low interval rates.

[3] It is however well know that the conflict between Bayesian and frequentist procedures tends to disappear as the sample size increases. Indeed, the discrepancies are limited when sampling information dominates the prior distribution or pre-experimental information may influence the estimates on prior distribution.

Following a Bayesian approach, we consider H an unknown hypothesis; $\mathbf{X} = \{X_1, ..., X_n\}$ is a set of independent and identically distributed observations. Let $x_n = (x_1, ..., x_n)$ be an observed sample; $\pi(H)$ is the prior probability of the hypothesis under test; $\pi(\mathbf{X}|H)$ the likelihood; and the posterior distribution is defined as in equation 2.

$$\pi(H|\mathbf{X}) = \frac{\pi(\mathbf{X}|H)\pi(H)}{\sum_n \pi(\mathbf{X}_n|H_n)\pi(H_n)} \tag{2}$$

Predictive modeling involves finding good subsets of predictors or explanatory variables. Models that fit the data well are better than models that fit the data poorly. Simple models are better than complex models. Working with a list of useful predictors, we can fit many models to the available data, then evaluate those models by their simplicity and by how well they fit the data.

7 A Preliminary Example

To illustrate the approach proposed in KITE.it, we now provide a running example. Let us start from a sample business rule stating that: "On an equipment fault, operators will visit customers premises within 12 hours from fault reporting". Our aim is to discover new knowledge from the information detected by monitoring the process in connection to this policy. We then formulate a *predictive analysis* considering the incidence of "previous visits to the same client by the same operator" to violations to these policies.

We start by a *descriptive metrics* that can be computed using a query listing the *excess_time*, expressed in hours and computed as the difference between *visit_time* and *fault_time*, for a set of tuples extracted from specific traces identified by *ProcessID*. Table 8 illustrate an sample of the results returned querying a data set.

ProcessID	FaultID	VisitID	OpId	ExcessTime
12	AF01	AEFF	1	20
31	AB00	AB07	3	3
33	A777	AA01	7	16
15	AB43	AA08	4	4
39	A605	AAB0	9	8
29	AK15	AA04	13	19
11	AG33	AA42	14	7
21	AB06	AB17	8	14
11	AG43	AA22	12	16
05	AB23	AA78	19	13

Table 8. Excess time from equipment fault to visit of an operator.

The *prescription* we want to apply to these traces imposes a constraint of form $excess\_time > 12$. Filtering traces by this constraint we obtain the set of violations $V : \{12, 33, 29, 21, 11, 05\}$. This set must be compared to the set of traces ordered by the

the number of previous visits by same operator to clients. Another *descriptive metrics* is then defined to extract these data, getting a distribution E. Table 9 illustrate an sample of the results returned.

ClientID	VisitID	OpId	VisitPriorToFault
C121	AEFF	1	3
C313	AB07	3	2
C236	AA01	7	6
C118	AA08	4	4
C259	AAB0	9	1
C329	AA04	13	1
C311	AA42	14	2
C111	AB17	8	4
C319	AA22	12	6
C209	AA78	19	3

Table 9. Visit prior to fault from the operators involved in visits listed in Table 8.

The *predictive analysis* is then executed by evaluating the incidence of different partition E on V. More specifically, referring to the equation 2, V is the hypotheses H we are testing and E is the observation $\mathbf{X}$. We are in other words evaluating how confident we are that observing a trace included in E this trace will also be in V.

If $\mathbf{X}$ is an ordinal variable we can test these incidence for each subset of the distribution by imposing a threshold α for defining membership of the subset under consideration.

$$\mathbf{X}_\alpha = \{x \geq \alpha, \forall x \in \mathbf{X}\} \tag{3}$$

So we can straightforwardly proceed to calculate the posterior probability $\pi(V|\mathbf{E}_\alpha)$. For instance taking $\alpha = 3$ we have six process instances in $\mathbf{E}_\alpha$, with five of them in V: $\pi(V|\mathbf{E}_\alpha) = \frac{5}{6} = 0.83$. Table 10 shows the results imposing a thresholds α for each value in E.

α	Posterior Probability	Prior Probability
1	0.6	0.6
2	0.625	0.6
3	0.83	0.6
4	0.75	0.6
6	1	0.6

Table 10. Incidence to violations of different α on values of “visit prior to fault”.

We find that process instances related to 3 or more “visit prior to fault time” present high probability to violate the business rules defining the expected execs time from fault to visit. In particular this is shaping a behavior that is potential dysfunctional, e.g. due to a “cry wolf” effect.

It is important to note that in this example we used “educated guesses” to decide the set of parameters to be used for the process behavior metrics. An exhaustive search of the right parameter set to identifying inductive metrics would be computationally very expensive. Clearly several not exhaustive approaches are possible. Ranging from considering the expert intuitions to game-theoretical algorithm aimed at identifying parameter sets based on their effectiveness in the winning strategy for an attacker wishing to fail the KPI without being caught, as explained in [31].

8 Conclusion

Most of the time, the literature has disregarded a notion of process behaviour that comprehensively includes alla data related to resources auxiliary to process execution. As a consequence, the method proposed for implementing Predictive Analytics usually are not fully integrated with a Knowledge Acquisition procedure, for instance, without providing concrete guidelines on how to move form one measurement step to another.

In this paper we put forward the idea that the full integration of PI capabilities requires to introduce a notion of Extended Behaviour, as the value of the information available in processes becomes one of the most important source for Predictive Analytics bringing to the acquisition of novel knowledge. .

Acknowledgment

This work was partly funded by the Italian Ministry of Economic Development under the “Industria 2015” contract - KITE.IT Project.

References

1. Surajit, C., Umeshwar, D., Vivek, N.: An overview of business intelligence technology. Commun. ACM **54**(8) (2011) 88–98
2. Yoo, Y.S., Yu, J., Lee, B.B., Bang, H.C.: A study on ubiquitous business process management for real-time open usn mash-up services. In: Information Technology Convergence, Secure and Trust Computing, and Data Management. Springer (2012) 21–27
3. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Process intelligence. In: Fundamentals of Business Process Management. Springer (2013) 353–383
4. Colombo, A., Damiani, E., Frati, F., Oltolina, S., Reed, K., Ruffatti, G.: The use of a metamodel to support multi-project process measurement. In: Proceedings of 15th Asia-Pacific software engineering conference (APSEC 2008), Beijing, China (2008) 503–510
5. Arigliano, F., Bianchini, D., Cappiello, C., Corallo, A., Ceravolo, P., Damiani, E., De Antonellis, V., Pernici, B., Plebani, P., Storelli, D., et al.: Monitoring business processes in the networked enterprise. In: Data-Driven Process Discovery and Analysis. Springer (2012) 21–38
6. Azzini, A., Ceravolo, P., Damiani, E., Zavatarelli, F., Vicari, C., Savarino, V.: Driving knowledge acquisition via metric life-cycle in process intelligence. In: Proceedings of the 14th International Conference on Knowledge Technologies and Data-driven Business, ACM (2014) 26

7. Ceravolo, P., Zavatarelli, F.: Knowledge acquisition in process intelligence. In: Proceedings of the International Conference on Information and Communication Technology Research. (to be pblished)
8. Maggi, F.M., Francescomarino, C.D., Dumas, M., Ghidini, C.: Predictive monitoring of business processes. In: CAiSE. (2014) 457–472
9. van der Aalst, W., Schonenberg, M., Song, M.: Time prediction based on process mining. Information Systems **36**(2) (2011) 450 – 475 Special Issue: Semantic Integration of Data, Multimedia, and Services.
10. Kang, B., Kim, D., Kang, S.H.: Real-time business process monitoring method for prediction of abnormal termination using knni-based lof prediction. Expert Syst. Appl. **39**(5) (April 2012) 6061–6068
11. Folino, F., Guarascio, M., Pontieri, L.: Discovering context-aware models for predicting business process performances. In: On the Move to Meaningful Internet Systems: OTM 2012. Springer (2012) 287–304
12. Pika, A., van der Aalst, W.M., Fidge, C.J., ter Hofstede, A.H., Wynn, M.T.: Predicting deadline transgressions using event logs. In: Business Process Management Workshops, Springer (2013) 211–216
13. Suriadi, S., Ouyang, C., van der Aalst, W.M., ter Hofstede, A.H.: Root cause analysis with enriched process logs. In: Business Process Management Workshops, Springer (2013) 174–186
14. Papazoglou, M., Heuvel, W.V.D.: Business process development life cycle methodology. In: Communications of the ACM. (2007) 79–85
15. Li, M., Liu, L., Yin, L., Zhu, Y.: A process mining based approach to knowledge maintenance. Information Systems Frontiers **13**(3) (2011) 371–380
16. Arigliano, F., Azzini, A., Braghin, C., Caforio, A., Ceravolo, P., Damiani, E., Savarino, V., Vicari, C., Zavatarelli, F.: Knowledge and business intelligence technologies in cross-enterprise environments for italian advanced mechanical industry. In: Proceedings of the 3rd International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2013), Riva del Garda (TN), CEUR-WS.org (2013) 104–110
17. Hayes, P., McBride, B.: Resource description framework (rdf). `http://www.w3.org/standards/techs/rdf` Date: 2014.
18. Carroll, J., Bizer, C., Hayes, P., Stickler, P.: Named graphs. Journal of Web Semantics **3**(3) (2005)
19. Garlik, S.H., Seaborne, A.: Sparql 1.1 query language. `http://www.w3.org/TR/2013/REC\--sparql11\--query\--20130321/` Date: 2013.
20. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of sparql. ACM Transactions on Database Systems (TODS) **34**(3) (2009) 16
21. Leida, M., Majeed, B., Colombo, M., Chu, A.: Lightweight rdf data model for business processes analysis. Data-Driven Process Discovery and Analysis Series: Lecture Notes in Business Information Processing **116** (2012)
22. Holsapple, C.W.: The inseparability of modern knowledge management and computer-based technology. Journal of knowledge management **9**(1) (2005) 42–52
23. Arigliano, F., Bianchini, D., Cappiello, C., Corallo, A., Ceravolo, P., Damiani, E., Antonellis, V.D., Pernici, B., Plebani, P., Storelli, D., Vicari, C. Lecture notes in business information processing ; 116. In: Monitoring business processes in the networked enterprise. Springer, Berlin (2012)
24. Von Halle, B., Goldberg, L.: Business Rule Revolution (ebook): Running Business the Right Way. Happy About (2006)
25. Taylor, J.: Decision Management Systems: A Practical Guide to Using Business Rules and Predictive Analytics. Pearson Education (2011)

26. Bezerra, F., Wainer, J., van der Aalst, W.M.: Anomaly detection using process mining. In: Enterprise, Business-Process and Information Systems Modeling. Springer (2009) 149–161
27. Wombacher, A., Li, C.: Alternative approaches for workflow similarity. In: Services Computing (SCC), 2010 IEEE International Conference on, IEEE (2010) 337–345
28. Van Der Aalst, W., Van Hee, K.: Workflow management: models, methods, and systems. MIT press (2004)
29. Fanning, K., Centers, D.P.: Intelligent business process management: Hype or reality? Journal of Corporate Accounting & Finance **24**(5) (2013) 9–14
30. Lee, P.M.: Bayesian statistics: an introduction. John Wiley & Sons (2012)
31. Tomas, C.J.: Game theory methods of identifying potential key factors. https://people.stanford.edu/calebj/content/game-theory-methods-identifying-potential-key-factors-0 (2012)

Compact Regions for Place/Transition Nets

Robin Bergenthum

Department of Software Engineering and Theory of Programming,
FernUniversität in Hagen
robin.bergenthum@fernuni-hagen.de

Abstract. This paper presents compact regions to synthesize a Petri net from a partial language. We synthesize a Petri net using the theory of regions. Let there be a partial language, every region definition provides an inequality system and a solution of this system is called a region. Every region defines a valid place where a place is valid if it enables every word of the partial language. The new compact region definition relies on compact tokenflows. Compact tokenflows are a very efficient behavioral model for the partial language of Petri nets [3, 4]. Compact regions will lead to faster synthesis algorithms computing smaller Petri nets solving the synthesis problem.

1 Introduction

We synthesize a place/transition net (p/t-net) from a partial language, i.e. a set of labeled partial orders (lpos) using the theory of regions [1, 2, 5]. Every type of region is based on a behavioral model. Of course, different behavioral models result in different region definitions, but the concepts in language based region theory are always the same [6, 7].

A place of a Petri net is valid for a specified language if it enables all words of the language. If we execute the language in the net, the firing rule is always satisfied for every valid place. Fix some type of behavioral model, fix a place, and fix a language: there is an inequality system so that there is a solution of this system if and only if the place is valid. The main idea of region theory is to consider the place as a variable in this inequality system. All at once, we are able to solve the synthesis problem, because we are able to calculate the set of all valid places.

The main steps of a region based synthesis algorithm are: Let there be a language over a set of labels and build a transition for every label. Choose a behavioral model and build the corresponding inequality system to check if an arbitrary place is valid. Consider the place to be a variable and calculate the basis of the solution space. For every solution in the basis add a place (and its arcs) to the set of transitions. The resulting net solves the synthesis problem, because of two arguments: First, every place of the constructed net is valid so that the Petri net enables all words of the language. Second, every additional place (not in the net) is either a linear combination of added places or is not valid. Altogether, the constructed net acts as specified or there is no such net.

In this synthesis algorithm, different behavioral models result in different inequality systems describing valid places. The run-time, as well as the size of the calculated Petri net, are mainly determined by the size of this inequality system. In the literature, there are two different types of regions considering partial order behavior of Petri nets. Both definitions have plenty disadvantages in most examples and applications. A transition region [7] belongs to the behavioral model of enabled cuts [8]. A place is valid if every set of unordered events of a specified language is enabled to occur after the occurrence of its prefix. The enabled cuts inequality system states that every prefix (i.e. its Parikh vector plus initial marking) produces enough tokens to enable the next maximal step. Therefore, transition regions yield an inequality for every cut of a language. A tokenflow region [6] belongs to the behavioral model of tokenflows [9]. A place is valid if there is a valid distribution of tokens along the transitive order relation of the specified language. Such a distribution of tokens needs to respect the firing rule of Petri nets. The tokenflow inequality system demands that every event does not produce too many tokens and every event receives enough tokens to occur. Therefore, there is a variable for every arc of the specified language as well as two inequalities for every event.

In practical applications, partial languages have many cuts and many arcs. The enabled cuts inequality system as well as the tokenflow inequality system is huge. Therefore, both existing synthesis algorithms have impracticable run-time. Paper [7] states, the more concurrency is in the language the worse is a transition region algorithm. The more dependency is in the language the worse is a tokenflow region algorithm.

In this short paper we present a new definition of compact region. This definition is related to the most recent behavioral model for partial languages presented in [3, 4]. We will show that compact regions lead to much smaller region inequality systems. The size of these systems is related to the size of the Hasse-diagrams of the specified language. Furthermore, the reduced size leads to smaller, i.e. nicer, Petri nets.

2 Preliminaries

Let f be a function and B be a subset of its domain A. We write $f|_B$ to denote the restriction of f to B. A function $m : A \to \mathbb{N}$ is called a multiset. We write $m = \sum_{a \in A} m(a) \cdot a$ to denote multiplicities of elements in m. Let $m' : A \to \mathbb{N}$ be another multiset. We write $m \geq m'$ if $\forall a \in A : m(a) \geq m'(a)$ holds. We denote the transitive closure of an acyclic and finite relation $\prec$ by $\prec^*$. We denote the skeleton of $\prec$ by $\prec^\diamond$. The skeleton of $\prec$ is the smallest relation $\lhd$ so that $\lhd^* = \prec^*$ holds. We model concurrent or distributed systems by place/transition nets [10].

Definition 1. *A place/transition net (p/t-net) is a tuple (P, T, W) where P is a finite set of places, T is a finite set of transitions so that $P \cap T = \emptyset$ holds, and $W : (P \times T) \cup (T \times P) \to \mathbb{N}$ is a multiset of arcs. A marking of (P, T, W) is a multiset $m : P \to \mathbb{N}$. Let m_0 be a marking. We call $N = (P, T, W, m_0)$ a marked p/t-net and m_0 the initial marking of N.*

Let t be a transition. We denote $\circ t = \sum_{p \in P} W(p, t) \cdot p$ the weighted preset of t. We denote $t\circ = \sum_{p \in P} W(t, p) \cdot p$ the weighted postset of t. A transition t can fire at marking m if $m \geq \circ t$ holds. If t fires, m changes to $m' = m - \circ t + t\circ$. The most famous behavioral model for partial order behavior of Petri nets is a process [11]. A process is a Petri net modeling one single run of a marked p/t-net.

Definition 2. *A process net is a tuple $O = (C, E, F)$ where C is a finite set of conditions, E is a finite set of events so that $C \cap E = \emptyset$ holds, and $F \subset (C \times E) \cup (E \times C)$*

is a set of arcs. The graph $(C \cup E, F)$ *is acyclic and does not branch at conditions, i.e. every condition has at most one predecessor and at most one successor.*

Let $N = (P, T, W, m_0)$ *be a marked p/t-net. A process is a pair* (O, ρ) *where* $O = (C, E, F)$ *is a process net and* $\rho : (C \cup E) \to (P \cup T)$ *is a labeling function.* (O, ρ) *is a process of* N *iff the following conditions hold:*

(1) $\rho(C) \subseteq P$, $\rho(E) \subseteq T$,

(2) $\forall e \in E : \circ\rho(e) = \sum_{p \in P} |\{(c, e) \in F | \rho(c) = p\}| \cdot p$,

(3) $\forall e \in E : \rho(e)\circ = \sum_{p \in P} |\{(e, c) \in F | \rho(c) = p\}| \cdot p$,

(4) $\forall p \in P : |\{c \in C | \forall e \in E : (e, c) \notin F, \rho(c) = p\}| = m_0(p)$.

Every process of a Petri net N relates to a labeled partial order. The set of labeled partial orders induced by all processes of N is the partial language of N.

Definition 3. *A labeled partial order (lpo) is a triple* $lpo = (V, <, l)$ *where* V *is a finite set of events,* $< \subseteq V \times V$ *is a transitive and irreflexive relation, and the labeling function* $l : V \to T$ *assigns a label to every event.*

Definition 4. *Let* $K = (C, E, F, \rho)$ *be a process of a marked p/t-net* N. *The lpo* $(E, F^*|_{E \times E}, \rho|_E)$ *is the process lpo of* K. *Let* N *be a marked p/t-net and* $L^\Pi(N)$ *be the set of all process lpos of* N. $L(N) = \{(E, <, l) | (E, <, l)$ *an lpo,* $(E, \prec, l) \in L^\Pi(N), \prec \subseteq <\}$ *is the partial language of* N.

We specify behavior of a system by example runs. An example run is a set of events with an acyclic relation. Of course, every example run relates to an lpo and we can extend the partial language of Petri nets to example runs.

Definition 5. *A triple* $run = (V, \prec, l)$ *is an example run if* $(V, \prec^*, l)$ *is a labeled partial order. A finite set of example runs is a specification. Let* $run = (V, \prec, l)$ *be an example run. We define* $run^* = (V, \prec^*, l)$ *the lpo and* $run^\diamond = (V, \prec^\diamond, l)$ *the compact lpo (cpo) of* run.

Definition 6. *Let* N *be a marked p/t-net and* $S = \{run_1, \ldots, run_n\}$ *be a specification. We write* $S \subseteq L(N)$ *iff* $\{run_1^*, \ldots, run_n^*\} \subseteq L(N)$ *holds.*

Synthesis is to construct a Petri net so that its behavior matches the specification. If there is no such net, we construct a net so that its behavior includes the specification and has minimal additional behavior.

Definition 7. *Let* S *be a specification, the synthesis problem is to compute a marked p/t-net* N *so that the following conditions hold:* $S \subseteq L(N)$ *and for all marked p/t-nets* $N' : L(N) \backslash L(N') \neq \emptyset \Longrightarrow S \not\subseteq L(N')$.

3 Compact Regions

We synthesize a p/t-net from a partial language applying the theory of regions. We construct a transition for every label of the specification and get a p/t-net without places. The language of this net includes arbitrary behavior. Obviously, we need to add places and arcs to restrict the behavior of such an initial net. Of course, we only add places that do not prohibit the specification.

Definition 8. *Let S be a specification and $N = (P, T, W, m_0)$ be a marked p/t-net. A place $p \in P$ is called feasible for S iff $S \subseteq L((\{p\}, T, W|_{(\{p\} \times T) \cup (T \times \{p\})}, m_0(p)))$.*

Let S be a specification and $N = (\{p\}, T, W, m_0)$ be a marked one-place p/t-net. We call N a feasible p/t-net for S if p is feasible for S.

Corollary 1. *Let S be a specification and T be its set of labels. Let $\{(\{p_1\}, T, W_1, m_1), \ldots, (\{p_n\}, T, W_n, m_n)\}$ be a set of feasible p/t-nets and $N = (\bigcup_i p_i, T, \bigcup_i W_i, \bigcup_i m_i)$ be their union. Of course, every place of N is feasible and $S \subseteq L(N)$ holds.*

In region theory it is well known that the following theorem holds [5].

Theorem 1. *Let S be a specification and T be its set of labels. The infinite p/t-net which is the union of all feasible p/t-nets is a solution of the synthesis problem.*

To discover feasible places, we use compact tokenflows as behavioral model [3, 4].

Definition 9. *Let $N = (P, T, W, m_0)$ be a p/t-net and $run = (V, \prec, l)$ be an example run so that $l(V) \subseteq T$ holds. We denote $\prec^\diamond = \lhd$. A compact tokenflow is a function $x : (V \cup \lhd) \to \mathbb{N}$. x is compact valid for $p \in P$ iff the following conditions hold:*

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

(ii) $\forall\, v \in V$: $\sum_{v \lhd v'} x(v, v') \leq x(v) + \sum_{v' \lhd v} x(v', v) - W(p, l(v)) + W(l(v), p)$,

(iii) $\sum_{v \in V} x(v) \leq m_0(p)$.

run is compact valid for N iff there is a compact valid tokenflow for every $p \in P$.

Papers [3, 4] prove that the partial language of a marked p/t-net is the set of its compact valid example runs.

Theorem 2. *The language of a marked p/t-net is its set of compact valid example runs.*

At this point we are able to state the main contribution of this short paper. We take advantage of compact tokenflows and define the notion of a compact region.

Definition 10. *Let $S = \{(V_1, \prec_1, l_1), \ldots, (V_n, \prec_n, l_n)\}$ be a specification, T be its set of labels, and p be a place. We denote $\prec_i^\diamond = \lhd_i$.*

A function $r : (\bigcup_i (V_i \cup \lhd_i) \cup (T \times \{p\}) \cup (\{p\} \times T) \cup \{p\}) \to \mathbb{N}$ is a compact region for S iff $\forall i \in \mathbb{N} : r|_{\{V_i \cup \lhd_i\}}$ is compact valid for p in $(\{p\}, T, r|_{(T \times \{p\}) \cup (\{p\} \times T)}, r(p))$.

Theorem 3. *Let S be a specification and T be its set of labels. Every compact region r for S defines a feasible p/t-net $N_r = (\{p\}, T, W, m_0)$ and vice versa.*

Proof. Let r be a compact region. For every example run in S there is a valid compact tokenflow $r|_{\{V_i \cup \lhd_i\}}$ of p in $N_r = (\{p\}, T, r|_{(T \times \{p\}) \cup (\{p\} \times T)}, r(p))$. Of course, $S \subseteq L(N_r)$ holds and N_r is feasible.
Let $N = (\{p\}, T, W, m_0)$ be a feasible p/t-net so that $S \subseteq L(N_r)$ holds. There is a valid compact tokenflow r_i for every example run of S. Obviously, the union $r = \bigcup_i r_i \cup W \cup m_0$ is a compact region.

Altogether, we are able to express the set of all feasible p/t-nets by a single inequality system. In this system there is a variable for every element in the domain of a compact region, i.e. one variable for every event, another variable for every skeleton arc, two variables for every label, and a single variable for the initial marking. The complete inequality system is built from the inequalities defined in Definition 9. According to (i) and (ii) there are two inequalities for every event of the specification. According to (iii) there is another inequality for every example run. The set of positive integer solutions of this inequality system is the set of all feasible nets. We call this inequality system the compact region inequality system. The union of positive integer basis solutions of the compact region inequality system is a solution of the synthesis problem.

The transition region inequality system [7] and the tokenflow region inequality system [6] are defined just like a compact region inequality system. The tokenflow region inequality system has two additional variables for every transitive arc of the specification. For most examples the size of the tokenflow system is quadratic in the size of the compact system. The transition region inequality system has one inequality for every cut of the specification. Note, the number of cuts may be exponential in the number of events and the number of cuts is always bigger than the number of skeleton arcs. Altogether, the compact system is always smaller than the other two inequality systems. Compact regions are the most efficient definition of a region of a partial language. Right now, I am implementing compact region synthesis in a tool called MoPeBs Alpaca. First results are very promising and match the theoretical considerations. Synthesis is much faster and the compact solution space leads to nets having fewer places. Of course, this is provisional. Comprehensive implementation and run-time tests is future work.

References

[1] Ehrenfeucht, A.;Rozenberg, G.: *Partial (Set) 2-Structures. Part I: Basic Notions and the Representation Problem*. Acta Inf. 27 (4), 315–342, 1990.

[2] Ehrenfeucht, A.;Rozenberg, G.: *Partial (Set) 2-Structures. Part II: State Spaces of Concurrent Systems*. Acta Inf. 27 (4), 343–368, 1990.

[3] Bergenthum, R.: *Verifikation von halbgeordneten Abläufen in Petrinetzen*. PhD in computer science, Library of the University of Hagen, 2013.

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

[5] Badouel, E.; Darondeau P.: *Theory of Regions*. Lectures on Petri Nets, LNCS 1491, 529–586, Springer, 1998.

[6] Bergenthum, R.; Desel, J.; Lorenz, R.; Mauser, S.: *Synthesis of Petri Nets from Finite Partial Languages*. Fundamenta Informaticae 88, 437–468, IOS Press, 2008.

[7] Bergenthum, R.; Desel, J.; Mauser, S.: *Comparison of Different Algorithms to Synthesize a Petri Net from a Partial Language*. ToPNoC 3, LNCS 5800, 216–243, Springer, 2009.

[8] Grabowski, J.: *On partial languages*. Fundamenta Informaticae 4, 427–498, IOS Press, 1981.

[9] Bergenthum, R.; Desel, J.; Juhs, G.; Lorenz, R.; Mauser, S.: *Executability of Scenarios in Petri Nets.*. Theoretical Computer Science 410 (12-13), 1190–1216, Elsevier, 2009.

[10] Desel, J.; Reisig, W.: *Place/Transition Petri Nets*. Lectures on Petri Nets, LNCS 1491, 122–173, Springer, 1998.

[11] Goltz, U.; Reisig, W.: *Processes of Place/Transition-Nets*. Automata Languages and Programming 154, 264–277, Springer, 1983.

An Optimal Process Model for a Real Time Process

Likewin Thomas*, Manoj Kumar M V*, Annappa B*, and Vishwanath K P†

*Department of Computer Science and Engineering
†Department of Mathematical and Computational Science
National Institute of Technology Karnataka, Surathkal,
Mangalore - 575025
India
{likewinthomas, manojmv}@nitk.ac.in
annappa@ieee.org
shastryvishwanath@gmail.com
http://www.cse.nitk.ac.in

Abstract. Recommending an optimal path of execution and a complete process model for a real time partial trace of large and complex organization is a challenge. The proposed AlfyMiner ($\alpha_y Miner$) does this recommendation in cross organization process mining technique by comparing the variants of same process encountered in different organization. $\alpha_y Miner$ proposes two novel techniques *Process Model Comparator* ($\alpha_{y Comp}$) and *Resource Behaviour Analyser* (RBA_{Miner}). $\alpha_{y Comp}$ identifies Next Probable Activity of the partial trace along with the complete process model of the partial trace. RBA_{Miner} identifies the resources preferable for performing Next Probable Activity and analyse their behaviour based on *performance, load and queue.* $\alpha_y Miner$ does this analysis and recommend the best suitable resource for performing Next Probable Activity and process models for the real time partial trace. Experiments were conducted on process logs of CoSeLoG Project[1] and *72%* of accuracy is obtained in identifying and recommending *NPA* and the performance of resources were optimized by *59%* by decreasing their load.

Keywords: Cross Organization Process Mining, Resource Behavior, Best Resource, Polynomial Regression Model, Resource Performance, Resource Load, Resource Queue: Average Waiting Time.

1 Introduction

In the current world where the resources are being shared among different organization through the cloud computing paradigm, most of the organizations have started to shift towards Shared Business Process Management Infrastructure (SBPMI). Due to this shift in modelling paradigm, organizations have to

[1] http://dx.doi.org/10.4121/uuid:26aba40d-8b2d-435b-b5af-6d4bfbd7a270

continuously improve their process [1]. But most of the organizations are still depending on the external service providers to monitor their business process, hence the business links are to be established with those external agencies [2]. This issue was well addressed by the Information Technology by developing various work-flow tools [3] [4] [5] [6]. The challenge here is to extend the service from boundary of *single organization to cross organizations.*

Due to *data explosion* [7] getting insight and performing analysis on the data to understand their behaviour and discover an optimized process model is always been a challenge to any organization in the process mining environment. $\alpha_y Miner$ uses SBPMI, to analyse the data behaviour of an organization. This is achieved by comparing the model of same variant using RBA$_{Miner}$ in SBPMI and recommending the best suitable process model. The context of this paper is the CoSeLoG Project[2]. The data used for the experiment and analysis of proposed algorithm is obtained from the Configurable Services for Local Government (CoSeLoG) Project. This project was executed under Dutch Organization for Scientific Research (NWO) [8].

$\alpha_y Miner$ is a new analytical tool for discovering the optimal path of completion of a partial trace along with recommendation of complete process model. It proposes two novel techniques $\alpha_{y\,Comp}$ and RBA$_{Miner}$. $\alpha_{y\,Comp}$ identifies the optimal path of completion by matching the partial trace and discovering the variants in all process models logged in the repository. It identify and recommends the Next Probable Activity (NPA) of partial trace. RBA$_{Miner}$ identifies the suitable resource for performing the discovered NPA, by analysing the behaviour of all resources capable of performing NPA based on their *performance, load and waiting time.*

$\alpha_y Miner$ is analysed using the running example [2]. NPA for the partial trace and optimal process model is identified in cross organization environment using $\alpha_{y\,Comp}$ [3] and the resource preferable for performing NPA is analysed and recommended using RBA$_{Miner}$ [4]. The experiment is conducted using the real time event log of CoSeLoG Project[3] and the result of RBA$_{Miner}$ is presented in section [5].

2 Running Example

The proposed $\alpha_y Miner$ is illustrated using the running example of four variant process model containing 9 activities, shown in Figure[1b]. The corresponding sample event log describing the process execution of the process model is shown in Table[1]. Here the traces matches model perfectly which is not the cases in real life process model. The complete log file of the running example can be

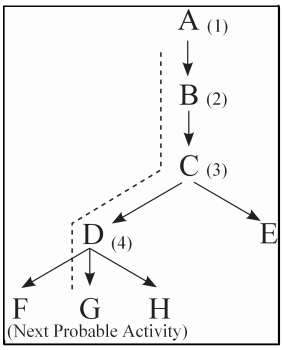
[2] http://dx.doi.org/10.4121/uuid:26aba40d-8b2d-435b-b5af-6d4bfbd7a270

[3] http://dx.doi.org/10.4121/uuid:26aba40d-8b2d-435b-b5af-6d4bfbd7a270

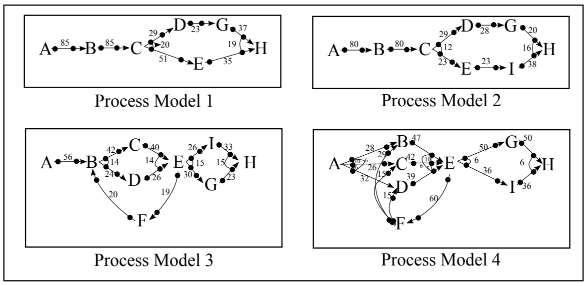
found at *Process Mining* @ *NITK*[4]. The experimental results are obtained using the CoSeLoG Project[5].

2.1 Proposed Problem

Consider an online process shown in Figure[1a], the dotted line shows the path of execution of the online process. Sub-scripted values at each activities are the sequence of occurrence of the activities ($A_1 \rightarrow B_2 \rightarrow C_3$). At activity C_3, decision has to be taken about which next activity to be performed, either D or E. $\alpha_y Miner$ identify the NPA and *recommends* the suitable resource for performing NPA.



(a) **Illustration of Online Process Model**



(b) **Process Models**: Four variants of interview process (*registration* (A), *validity check* (B), *document check* (C), *information check* (D), *decide* (E), *interview* (I), *group discussion* (G), *result* (H) and *re-initiates* (F))

Fig. 1: **Running Example**

3 Alfy Miner ($\boldsymbol{\alpha_y Miner}$)

$\alpha_y Miner$ is intended to identify and predict the optimal path of execution along with the complete process model, for a real time process. On identifying the currently executing activity A_i, $\alpha_y Miner$ recommends the optimal path of completion and the best suitable process model matching the *partial trace* with same variant event logs, logged in the process model repository. On identifying the matched variants, the optimal process models are identified by running *process model comparator* $\alpha_{y\,Comp}$ which matches the partial trace. Recommendation of *Next probable Activity* NPA is done by selecting NPA (A_i) in identified suitable process model. The Algorithm [1] gives the execution steps of $\alpha_y Miner$.

[4] http://http://processminingnitk.blogspot.in/2015/03/best-resource-recommendation-for.html

[5] http://dx.doi.org/10.4121/uuid:26aba40d-8b2d-435b-b5af-6d4bfbd7a270

Case ID	TRACE	Duration
10358444	$A^{12350}_{24/01/14}$ $B^{630640}_{28/01/14}$ $C^{221210}_{29/01/14}$ $D^{23640}_{02/02/14}$ $E^{7560}_{15/02/14}$ $H^{631250}_{26/02/14}$	33
12421232	$A^{23640}_{25/01/14}$ $B^{530640}_{26/01/14}$ $C^{230410}_{28/01/14}$ $D^{12350}_{09/02/14}$ $G^{7716}_{13/02/14}$ $H^{631250}_{24/02/14}$	30
12592056	$A^{12350}_{02/03/14}$ $B^{4503}_{12/03/14}$ $C^{630450}_{18/03/14}$ $G^{721560}_{26/03/14}$ $E^{7560}_{27/03/14}$ $H^{631250}_{08/04/14}$	37
12610928	$A^{23640}_{12/05/14}$ $B^{530640}_{17/05/14}$ $C^{230410}_{29/05/14}$ $E^{7560}_{05/06/14}$ $D^{23640}_{16/06/14}$ $G^{7716}_{28/06/14}$ $H^{631250}_{05/07/14}$	54
12984815	$A^{12350}_{16/08/14}$ $B^{630450}_{29/08/14}$ $C^{221210}_{09/09/14}$ $D^{12350}_{16/09/14}$ $G^{721560}_{22/09/14}$ $E^{7716}_{15/10/14}$ $H^{631250}_{27/10/14}$	72

(a) **Event Log of Process Model 1**

Case ID	TRACE	Duration
13945854	$A^{23640}_{26/01/14}$ $B^{450320}_{28/01/14}$ $C^{630450}_{31/01/14}$ $D^{23640}_{15/02/14}$ $G^{720560}_{19/02/14}$ $H^{631250}_{26/02/14}$	31
13968144	$A^{12350}_{12/02/14}$ $B^{630450}_{19/02/14}$ $C^{221210}_{22/02/14}$ $E^{12350}_{09/03/14}$ $I^{631210}_{26/03/14}$ $H^{631250}_{28/03/14}$	44
15073705	$A^{12350}_{12/04/14}$ $B^{530640}_{29/04/14}$ $C^{630450}_{02/05/14}$ $D^{12350}_{15/05/14}$ $G^{771620}_{19/05/14}$ $E^{720560}_{26/05/14}$ $H^{631250}_{08/06/14}$	57
16609162	$A^{23640}_{15/04/14}$ $B^{530640}_{19/04/14}$ $C^{230410}_{02/05/14}$ $E^{720560}_{15/05/14}$ $D^{23640}_{16/05/14}$ $G^{771620}_{18/05/14}$ $H^{631250}_{20/05/14}$	35
16789201	$A^{12350}_{19/06/14}$ $B^{630450}_{23/06/14}$ $C^{221210}_{29/06/14}$ $D^{23640}_{15/07/14}$ $G^{721560}_{27/07/14}$ $E^{771620}_{09/08/14}$ $I^{641210}_{16/08/14}$ $H^{631250}_{23/08/14}$	65

(b) **Event Log of Process Model 2**

Case ID	TRACE	Duration
16796450	$A^{12350}_{02/05/14}$ $B^{450320}_{23/05/14}$ $C^{630450}_{15/06/14}$ $E^{720560}_{19/06/14}$ $I^{651210}_{09/07/14}$ $H^{631250}_{27/07/14}$	86
17031584	$A^{23640}_{26/07/14}$ $B^{450320}_{15/08/14}$ $C^{221210}_{29/08/14}$ $E^{720560}_{12/09/14}$ $F^{720560}_{28/09-14}$ $B^{630450}_{13/10/14}$ $C^{221210}_{18/10/14}$ $E^{720560}_{22/10/14}$ $I^{651210}_{29/10/14}$ $H^{631250}_{30/10/14}$	96
17939005	$A^{12350}_{05/10/14}$ $B^{630450}_{13/10/14}$ $C^{630450}_{22/10/14}$ $E^{720560}_{29/10/14}$ $F^{720560}_{13/11/14}$ $B^{450320}_{19/11/14}$ $D^{23640}_{02/12/14}$ $E^{720560}_{06/12/14}$ $G^{720560}_{10/12/14}$ $H^{631250}_{24/12/14}$	80
19472044	$A^{23640}_{15/12/14}$ $B^{530640}_{19/12/14}$ $C^{630450}_{28/12/14}$ $E^{720560}_{03/01/15}$ $F^{720560}_{05/01/15}$ $B^{630450}_{16/01/15}$ $C^{230410}_{18/01/15}$ $E^{720560}_{22/01/15}$ $G^{721560}_{23/01/15}$ $I^{631210}_{28/01/15}$ $H^{631250}_{29/01/15}$	45
25845687	$A^{23640}_{12/11/14}$ $B^{530640}_{14/12/14}$ $C^{630450}_{19/12/14}$ $E^{720560}_{22/12/14}$ $G^{721560}_{27/12/14}$ $H^{631250}_{30/12/14}$	48

(c) **Event Log of Process Model 3**

Case ID	TRACE	Duration
19830478	$A^{12350}_{02/05/14}$ $B^{12350}_{02/05/14}$ $E^{12350}_{02/05/14}$ $G^{12350}_{02/05/14}$ $H^{12350}_{02/05/14}$	53
19834032	$A^{12350}_{02/05/14}$ $B^{12350}_{02/05/14}$ $E^{12350}_{02/05/14}$ $F^{12350}_{02/05/14}$ $C^{12350}_{02/05/14}$ $E^{12350}_{02/05/14}$ $G^{12350}_{02/05/14}$ $H^{12350}_{02/05/14}$	52
19836934	$A^{12350}_{02/05/14}$ $B^{12350}_{02/05/14}$ $C^{12350}_{02/05/14}$ $E^{12350}_{02/05/14}$ $F^{12350}_{02/05/14}$ $B^{12350}_{02/05/14}$ $E^{12350}_{02/05/14}$ $G^{12350}_{02/05/14}$ $H^{12350}_{02/05/14}$	59
19838656	$A^{12350}_{02/05/14}$ $D^{12350}_{02/05/14}$ $B^{12350}_{02/05/14}$ $E^{12350}_{02/05/14}$ $F^{12350}_{02/05/14}$ $B^{12350}_{02/05/14}$ $E^{12350}_{02/05/14}$ $G^{12350}_{02/05/14}$ $H^{12350}_{02/05/14}$	37
19844185	$A^{12350}_{02/05/14}$ $D^{12350}_{02/05/14}$ $C^{12350}_{02/05/14}$ $E^{12350}_{02/05/14}$ $F^{12350}_{02/05/14}$ $B^{12350}_{02/05/14}$ $E^{12350}_{02/05/14}$ $G^{12350}_{02/05/14}$ $H^{12350}_{02/05/14}$	29

(d) **Event Log of Process Model 4**

Table 1: **Event logs of four different process models of interview process** shown in figure[1b]. Each log table shows Case ID_1, Trace_2 and the total $\mathrm{duration}_3$. Each cell in trace, shows the activity of the trace, *Resource* (Superscripted) and the time of occurrence of that activity (sub-scripted).

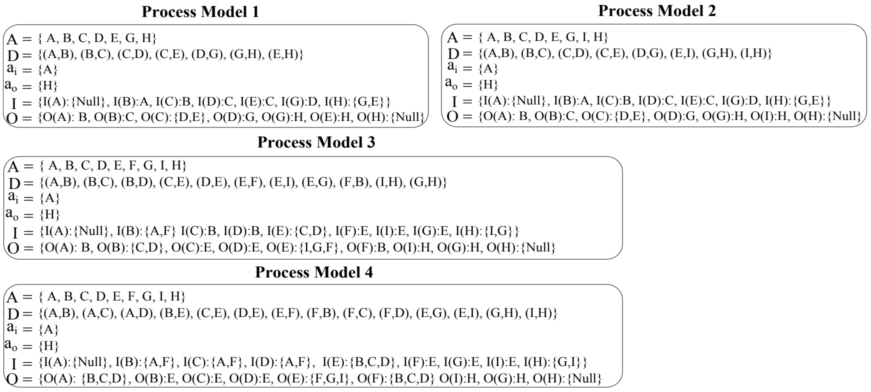
Algorithm 1: $\alpha_y Miner$

```
Input: Partial Real Time Trace
Output: NPA & Process Model
Develop Process model repository;
repeat
    $\mathrm{Match}_{Var}$ ← Call Match Variant($A_i$);
    $\alpha_y Comp$ ← $\alpha_y$ Comp($\mathrm{Match}_{Var}$) ;
    Set(NPA) ← $\mathbf{InOut}_{Binding}$(C-Net )
until for each currently executing activity $A_i$
```

3.1 Process Model: Casual Net

$\alpha_y Miner$ uses Casual Net: C-Net notation to represent the process model. C-Net is a six-tuple: $\{A,D,a_i,a_o,I,O\}$ representation of process model with A:{set of activities}, D:{Set of Dependencies}, a_i:{Set of Start activities}, a_o: {Set of Output activities} , I: {Set of Input Binding} , O: {Set of Output Binding}.

C-Net for all the four process model of the running example is shown in Figure 2. The repository of process model is maintained for analysing process behaviour.



Process Model 1

$A =$ { A, B, C, D, E, G, H}
$D =$ {(A,B), (B,C), (C,D), (C,E), (D,G), (G,H), (E,H)}
$a_i =$ {A}
$a_o =$ {H}
$I =$ {I(A):{Null}, I(B):A, I(C):B, I(D):C, I(E):C, I(G):D, I(H):{G,E}}
$O =$ {O(A): B, O(B):C, O(C):{D,E}, O(D):G, O(G):H, O(E):H, O(H):{Null}

Process Model 2

$A =$ { A, B, C, D, E, G, I, H}
$D =$ {(A,B), (B,C), (C,D), (C,E), (D,G), (E,I), (G,H), (I,H)}
$a_i =$ {A}
$a_o =$ {H}
$I =$ {I(A):{Null}, I(B):A, I(C):B, I(D):C, I(E):C, I(G):D, I(H):{G,E}}
$O =$ {O(A): B, O(B):C, O(C):{D,E}, O(D):G, O(G):H, O(I):H, O(H):{Null}

Process Model 3

$A =$ { A, B, C, D, E, F, G, I, H}
$D =$ {(A,B), (B,C), (B,D), (C,E), (D,E), (E,F), (E,I), (E,G), (F,B), (I,H), (G,H)}
$a_i =$ {A}
$a_o =$ {H}
$I =$ {I(A):{Null}, I(B):{A,F} I(C):B, I(D):B, I(E):{C,D}, I(F):E, I(I):E, I(G):E, I(H):{I,G}}
$O =$ {O(A): B, O(B):{C,D}, O(C):E, O(D):E, O(E):{I,G,F}, O(F):B, O(I):H, O(G):H, O(H):{Null}

Process Model 4

$A =$ { A, B, C, D, E, F, G, I, H}
$D =$ {(A,B), (A,C), (A,D), (B,E), (C,E), (D,E), (E,F), (F,B), (F,C), (F,D), (E,G), (E,I), (G,H), (I,H)}
$a_i =$ {A}
$a_o =$ {H}
$I =$ {I(A):{Null}, I(B):{A,F}, I(C):{A,F}, I(D):{A,F}, I(E):{B,C,D}, I(F):E, I(G):E, I(I):E, I(H):{G,I}}
$O =$ {O(A): {B,C,D}, O(B):E, O(C):E, O(D):E, O(E):{F,G,I}, O(F):{B,C,D} O(I):H, O(G):H, O(H):{Null}

Fig. 2: **C-Net Representation of process Model in Figure** 1b

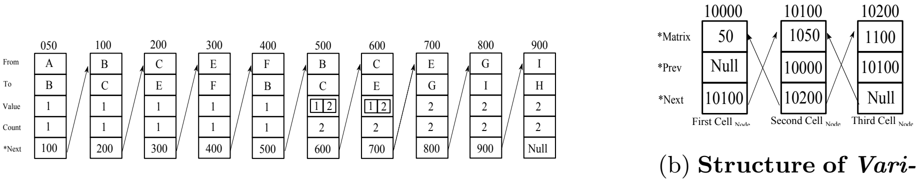
3.2 Matching variants with Path Detector

When an online process is getting executed, identifying to which variant the currently executing trace belongs is a challenge for $\alpha_y Miner$. Algorithm Variant $_{Match}$[2] identify the path of execution along with the set of possible NPA. Variant$_{Match}$ uses the concept of linked list with 2 nodes: *Cell*$_{Node}$ and *Variant*$_{Node}$ which are represented as *class*. *Cell*$_{Node}$ $= \{from_1 \leftarrow \bigcup\{\bullet a\}$, to$_2 \leftarrow a$, $value_3 \leftarrow \{|\bullet a \rightarrow a|\sigma\}$, $count_4 = |\bullet a \rightarrow a| \in \zeta$. *Variant*$_{Node}$ {$*matrix$ (address of *Cell*$_{Node}$), $*prev_2$ $*next_3$ (address of next and previous *Cell*$_{Node}$)}. The *Cell*$_{Node}$ Figure[3a] stores the information of trace A→B→C→E→F→B→D→E→G→H of process model 2. The *value*$_3$ field remains 1 till the sequence in trace appears first time. On identifying the *loop*, value in *value*$_3$ filed is updated to 2 as shown at *Cell*$_{Node}$ with memory 500 in Figure[3a]. *Value*$_3$ field is an array and stores the value 1,2 to indicate the sequence B→C is appearing second time in the trace.*Count*$_3$ is a counter of the sequence appearance in the trace. *Variant*$_{Node}$ Figure[3b] stores the information of all the variants. This is used while comparing the online sequence with the variants. *If* a variant matches the sequence, *then* that variant is retained *else* it is deleted from the linked list.

3.3 Process Model Comparator ($\alpha_{y\,Comp}$)

$\alpha_{y\,Comp}$ compares the C-Net of all the variants in cross organization environment based on following comparison metrics.

1. *Process Model Metric*: Compare total number of activities, resources, traces and variants
2. *Relation Metric*: Compare total number of parallel, serial activities and loops.



	050	100	200	300	400	500	600	700	800	900
From	A	B	C	E	F	B	C	E	G	I
To	B	C	E	F	B	C	E	G	I	H
Value	1	1	1	1	1	1 2	1 2	2	2	2
Count	1	1	1	1	1	2	2	2	2	2
*Next	100	200	300	400	500	600	700	800	900	Null

(a) **Structure of *Cell*$_{Node}$** for sequence A→B→C→E→F→B→D→E→G→H of process model 2

	10000	10100	10200
*Matrix	50	1050	1100
*Prev	Null	10000	10100
*Next	10100	10200	Null
	First Cell $_{Node}$	Second Cell $_{Node}$	Third Cell $_{Node}$

(b) **Structure of *Variant*$_{Node}$** for the set of *Cell*$_{Node}$ of process model 2

Fig. 3: **Structure of Cell$_{Node}$ and Variant$_{Node}$**

Algorithm 2: Matching the Variants: $Variant_{Match}()$

```
   Input: Online process
   Output: Matching matrix
1  Match Variant() struct variant$_{Node}$⋆gvn, ⋆tempvn; (gvn : address of linked list say
   globle Variant Node), Let ⋆gvn gives address of the double linked list, Initialize all counter
   in cell$_{Node}$ → 0;
2  repeat
3      ⋆tempvn ← &gvn Get the address of the double linked list;
4      repeat
5          ⋆tempcn ← &matrix Get the address of the matrix;
6          tempcn→from = sequence[i] ∧ tempcn→to = sequence[i+1];
7          if not found then Delete current variant$_{Node}$ from double linked list and go to 5
8          else Increment the member variable count;
9          if count == val[count] (Current and previous check are passed) then Go to
           next→variant$_{Node}$ in the double linked list and go to step 5
10         else Delete the current→variant$_{Node}$ from the double linked list and go to 5
11     until ⋆next in double linked list is null
12 until for each activity in online process
13 Remaining variant$_{node}$ present in tempvn are all matched variant table for the given
   sequence.
```

3. *Complexity Metric*: Compare total number of split and join.
4. *Service Time Metric*: Compare the queue time for each activity.
5. *Fitness Metric*: Running fitness test along with the time of completion and valid no of sequence in each event log.

Process Model Metric The process model comparison is done based on No of {Activities, Resources, Traces & Varinats } and is shown in Table 2a.

Relation Metric $\alpha_{y\,Comp}$ analysed that if a model has more parallel relation it performs well when compared to serial relation, at the same time if the loop is increased the consumption of execution time also increases. Parallel relation is identified by Equation 4 in Definition 1. Loops are identified by Equation 5.

Definition 1. ***Log based ordering relation***
Let $\mathscr{A}$ = [a, b, c, d, e] be the set of activities and let L be the simple event log

i.e., $L \in \mathscr{A}^{\ *}$ and Let A be a_i^{th} activity and B be a_{i+1}^{th} then,

$$\begin{aligned}
\dot{DirectlyFollow}_{(a >_L b)} &\leftarrow \{\textit{iff} \ \ \exists \ \ trace \ \ \sigma = \langle t_1, t_2, ..., t_n \rangle \wedge \\
&\qquad i \in [1, 2, ..., n-1] \mid \sigma \in L, \\
&\qquad \wedge \ \ t_i = a, \wedge \ t_i + 1 = b\} \qquad (1) \\
\dot{Casuality}_{(a \longrightarrow_L b)} &\leftarrow \{\textit{iff} \ \ a >_L b \wedge b \not>_L a\} \qquad (2) \\
\dot{Unrelated}_{(a \#_L b)} &\leftarrow \{\textit{iff} \ \ a \not>_L b \wedge b \not>_L a\} \qquad (3) \\
\dot{Parallel}_{(a \|_L b)} &\leftarrow \{\textit{iff} \ \ a >_L b \wedge b >_L a\} \qquad (4) \\
\dot{Loop}_{(a >_L b >_L a)} &\leftarrow \{\textit{iff}\, (a_i \ \ == \ \ a_{i+2}) \ \ \rightarrow \ \ a_i >_L a_{i+1} >_L a_{i+2}\} \qquad (5)
\end{aligned}$$

The Table 2b gives the relation metric of all the four models in running example.

Complexity Metric Complexity metric identifies the joins and splits in the process model. Joins and split are identified using the result of output and input binding. Consider the Figure[1b] where for process model 1: O(A)={B}=85 times, similarly the *split* {CDE} = 20, its means 20 times activity C is 20 times followed by both D and E, *join* {GEH} is joined 16 times. Using this information complexity metric shown in Table[2c] is developed.

Service Time Metric This metric gives the total service time comparison for an activity in each model. This comparison helps in identifying the model serving an activity with less service time. The service time is calculated by $\sum_{i=1}^{each\ cases} duration(A_i)$, where A$_i \subseteq \mathscr{A}(set\ of\ activities)$. The sample output in seconds is shown in Table 2d.

Fitness Metric This gives the numbers of traces that can be successfully run on the model. This is helpful in deciding how efficient the model is, in running the trace. $\alpha_{y\,Comp}$ identifies the model which runs maximum number of traces with minimum time. Consider the Table 2e.

3.4 Binding Relation

On identifying variants following the partial trace, the NPA of currently executing activity A$_i$ is identified using binding relation which bind the incoming and outgoing activity of A$_i$. Algorithm 3 eplain the concept of binding relation, where for each trace in a case, *if* an activity A is followed by B, *then* A.outbond ← B ∧ B.inbound ← A, i.e., A has *out-bounding relationship* with B and similarly B as *in-bounding relationship* with A

	No of Activities	No of Resources	No of Traces	No of Variants
PM 1	8	16	90	10
PM 2	8	14	80	13
PM 3	9	14	56	19
PM 4	9	14	86	51

(a) **Process Model Metric**

	No of Dependency	No of Parallel	No of Loops	No of Serial
PM1	7	2	0	5
PM2	8	4	0	4
PM3	11	2	2	7
PM4	14	3	3	

(b) **Relation Metric**

	Joins	Splits
PM1	19	20
PM2	16	12
PM3	29	29
PM4	33	28

(c) **Complexity Metric**

	A	B	C	D
PM1	3678956	45896374	56987845	1236589
PM2	2598964	56978746	78594785	4589647
PM3	4577896	36987567	23698124	5698347
PM4	1236978	23678945	22456378	4548768

(d) **Service Time Metric**

	PM1	T(PM1)	PM2	T(PM2)	PM3	T(PM3)	PM4	T(PM4)
Event Log1	1	56897845	0.9	78456975	0.75	45789647	0.65	56587874
Event Log2	0.8	45878123	1	45678412	0.9	78956478	0.95	78945698
Event Log3	0.6	45236984	0.75	56898774	1	69875457	1	65327841
Event Log4	0.45	32789564	0.6	68974564	0.75	39845641	1	

(e) **Fitness Metric**

Table 2: **Process Model Comparator** ($\alpha_{y\,Comp}$)

Algorithm 3: To calculate Input & Output Binding

```
InOut_Binding() Input: $A_i$, RTrace
Output: $A_i.Input_{Binding}$, $A_i.Output_{Binding}$
repeat
    if ($|a >_L b|$) then
        a.Outbound ← $b \wedge b.Inbound$ ← a
    $|a >_L b| = \sum_{\sigma \in L} L(\sigma) \times |\{1 \leq i < |\sigma| \mid \sigma(i) = a \wedge \sigma(i+1) = b\}|$ [see [7]]
until for each sequence in trace $\sigma$ in event log L
```

4 Resource Behaviour Analyser (RBA$_{Miner}$)

$\alpha_y Miner$ on discovering suitable process model with NPA identifies the resources preferable for performing NPA. Set of resource preferable for performing NPA is identified using Activity/Resource$_{rep}$[3]. RBA$_{Miner}$ analyse the behaviour and recommend the suitable resource for performing NPA. Behaviour of the resources is analysed based on 3 parameter: *Performance, Load and Queue* using polynomial regression model for load and performance [4.2] and Average Servicing Time at resource using queue model [4.3]. Algorithm 4 explains the concept of resource behaviour analysis.

4.1 Activity/Resource$_{rep}$

$\alpha_y Miner$ identifies the list of resources performing an activity in entire process log along with the time consumed by them for performing that activity. The Table 3 gives representational view of list of resources performing an activity in process model 1 along with the time consumed.

Algorithm 4: RBA$_{Miner}$

```
$RBA(NPA)()$
Input: $NPA\&BestRes_{Activity}$
Output: $Recommendationof Res_{(NPA)}$
repeat
    $Load(Res_{(NPA)}) \longleftarrow Poly.Load(Load(Res_{(NPA)}))$; [$see\ algo5$]
    $Perf(Res_{(NPA)}) \longleftarrow Poly.Perf(Res_{(NPA)})$; [$see\ algo5$]
    $AvgWaiting_Time(Res_{(NPA)}) \longleftarrow Queue(Res_{(NPA)})$; [$see\ algo$ 6]
until (for each resource of $NPA$ in $BestRes_{Activity}$ Table)
Recommend the optimal load, performance and waiting time resource
```

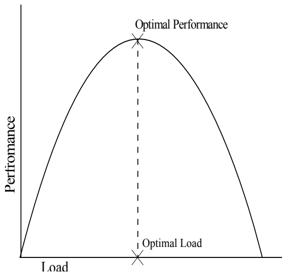
Activity	$\boldsymbol{Res}_{12350}$	$\boldsymbol{Res}_{23640}$	$\boldsymbol{Res}_{630450}$	$\boldsymbol{Res}_{530640}$	$\boldsymbol{Res}_{450320}$	$\boldsymbol{Res}_{221210}$	$\boldsymbol{Res}_{230410}$	$\boldsymbol{Res}_{501}$	$\boldsymbol{Res}_{771620}$	$\boldsymbol{Res}_{502}$	$\boldsymbol{Res}_{771620}$	$\boldsymbol{Res}_{721560}$
A	36.657	45.380	DNP	DNP	DNP	DNP	DNP	DNP	DNP	DNP	DNP	DNP
B	DNP	DNP	18.473	22.667	9.25	DNP	DNP	DNP	DNP	DNP	DNP	DNP
C	DNP	7	24.684	DNP	DNP	5.4667	22.294	DNP	DNP	DNP	DNP	DNP
D	DNP	25.53	DNP	DNP	DNP	DNP	DNP	72	11.5	DNP	DNP	DNP
E	DNP	25.531	DNP	DNP	DNP	DNP	DNP	62.5	DNP	91	11.5	DNP
G	DNP	DNP	DNP	DNP	DNP	DNP	DNP	DNP	7	DNP	DNP	13.944

Table 3: **Activity/Resource$_{rep}$ of process model 1 of running example** [DNP: *Did Not Play*]

4.2 Resource load & performance analyser

The *Yerkes-Dodson Law of Arousal*, also known as *Arousal Theory*, states that by increasing arousal, the workers performance can be improved. However, if the level of arousal increases too much, performance decreases Figure[4a] [9]. The RBA$_{Miner}$ identifies the *level of arousal*: *Optimal Load* i.e., the *maximum load* the resource can handle efficiently, along with its *performance* using *polynomial regression model.* Performance is a ratio of *Total time taken* by *Load.* The performance was analysed by increasing the load and observing the time taken. It was observed that, as the load was increased, the consumption of the time was decreasing. But at some point there was a drift and the time consumption started increasing. That drifted point is known as Arousal (optimal load and performance of the resources). The Algorithm[5] identifies the load ℓ and performance [$Total\ time \div \ell$] for /resource/unit time.

The RBA$_{Miner}$ first *filters* the *unperformed load*[1] (an activity with 0 ms) and *residual load*[2] (an activities with exceptional duration). Then the *actual load* (ℓ) and *average time of Service* (α) of each worker each month is identified. Polynomial regression model[5] is applied on this cleaned data. Since the RBA$_{Miner}$ is intended in identifying the second degree regression model, the regression model initialize a 3×3 matrix (A) and 3×1 matrix (B) as shown in figure [4b& 4c]. Then the transpose of matrix A is multiplied with matrix B. The result obtained is the coefficient of polynomial equation. On applying the load on an equation the polynomial curve (power curve) is obtained as shown in figure. On analysing the polynomial curve and applying the Yerkes-Dodson Law the *optimal load* and *optimal performance* of a resource is identified for each month.



$$A = \begin{bmatrix} n & \sum_{1}^{n}\ell & \sum_{1}^{n}\ell^2 \\ \sum_{1}^{n}\ell & \sum_{1}^{n}\ell^2 & \sum_{1}^{n}\ell^3 \\ \sum_{1}^{n}\ell^2 & \sum_{1}^{n}\ell^3 & \sum_{1}^{n}\ell^4 \end{bmatrix} \qquad B = \begin{bmatrix} \sum_{1}^{n}\alpha \\ \sum_{1}^{n}(\alpha\times\ell) \\ \sum_{1}^{n}(\alpha\times\ell^2) \end{bmatrix}$$

(a) **Yerkes Dodson Law** (b) **Matrix Table A** (c) **Matrix Table B**

Fig. 4: **Structure of Power Curve** for identifying the Optimal Load and Performance and the Structure Initial load & performance matrix for running Polynomial Regression Model

Algorithm 5: Resource *Second Order* Polynomial Regression Model

```
Input: $\ell$ (Total Load) on each resource each month and $\alpha$ (Log(Average Service Time)) for
       running the load $\ell$ per month
Output: Optimal Load $\mathscr{L}$ & Optimal Performance $\mathscr{P}$
Let A[3,3] & B[1,3] be 2 initial Matrix as shown in figure[4b & 4c]; k=0;
repeat
    $A^{Inverse} \longleftarrow Transpose(A,3)$; Transpose: Function transposing the matrix;
    $Result \longleftarrow multiplyMatrices(A^{Inverse}, B)$; multiplyMatrices: Function for
    multiplying matrix;
    repeat
        $\beta[i] \longleftarrow Result[i][j]$; where $\beta$= Coefficient of Polynomial Equation
    until ((i=0 to 3) $\wedge$ j=0)
    Polynomial Equation : $\beta_0 + \beta_1\ell + \beta_2\ell^2$
until (for each resource each unit time)
```

4.3 Activity Servicing Time Model

Along with identification of load and performance of the resource preferable for performing NPA, RBA$_{Miner}$ also finds the *Activity Servicing Time* (i.e., the average waiting time for an activity to be served by a resource), before that resource is recommended. Since the interest is in finding the queue at each resource, RBA$_{Miner}$ uses Single-Server Models *(M/M/1):(GD/∞/∞)* and *(M/M/1):(GD/N/∞)*. Here the model (M/M/1):(GD/∞/∞) describe *(Arrival*^{1}*/ Departure*^{2}*/ Server*^{3}*):(Queue discipline*^{4}*/ Max number in Queue*^{5}*/ Source of Calling*^{6}*)*.

Arrival^1 (λ) is the rate at which the activities are arrived at each resources and *Departure*^{2} (μ) is the rate at which the arrived activities are served. Since RBA$_{Miner}$ is intended in identifying the *average waiting time* at each resource, the single server model is applied. When data was analyzed for First Come First Serve *FCFS*, Last Come First Serve *LCFS* and Service in Random Order *SIRO*, it was understood that arrival of the activity was following General Discipline *GD* as its *Queue Discipline*^{4}. As the number in queue and source of calling is not defined RBA$_{Miner}$ marks them as *infinity.* The average waiting time in the system W_s is identified using Equations [6- 9]. The Algorithm Activity Servicing Time [6] starts with identifying the arrival rate λ and the servicing rate μ at each resources.

The λ_n & μ_n in generalized model is shown in Equation[6]. The traffic ρ: number of activities arriving and getting served per unit time is shown in Equation[7]. Hence the Average waiting time in system L_s is given in Equation[9].

$$\left.\begin{aligned} \lambda_n &= \lambda \\ \mu_n &= \mu \end{aligned}\right\} \quad \textit{Where } n = \textit{0,1,2....} \tag{6}$$

$$\rho = \frac{\lambda}{\mu} \tag{7}$$

$$W_s = \frac{L_s}{\lambda} \tag{8}$$

$$L_s = \frac{\rho}{1-\rho} \tag{9}$$

Algorithm 6: To Discover the Activity Servicing Time

```
Input: Set of resources:$\Re$, Trace:$\Im$, Duration of service:$\partial$
Output: Arrival $\lambda$, Service $\mu$, Traffic $\rho$, $L_s, W_s$
Let Arrival $\lambda \in$ Load $\ell$ discovered on /Resource/month; Service $\mu \in$ service rate of $\lambda$; $\mathit{\Pi}$ be No of Days in month
if (if $((\mathit{\Pi} - \Im.Date) \times 24hrs \times 60Sec) \geq \Im.\partial$ then Event is executed in same month;
    $\mu(\Re_{Filtered}\text{-}Year\_Month) \longleftarrow \mu(\Re_{Filtered}\text{-}Year\_Month) + 1$;
else $\perp = \lceil \frac{((\Im.\partial) - ((\mathit{\Pi} - \Im.Dt) \times 24hrs \times 60Sec))}{\mathit{\Pi} \times 24 \times 60} \rceil$
    $\mu(\Re_{Filtered}\text{-}Year\_Month + \perp) \longleftarrow \mu(\Re_{Filtered}\text{-}Year\_Month + \perp) + 1$;
Average Servicing Time in system $\leftarrow$ Equation [6 to 9]
```

5 Experimental Analysis and Result

The $\alpha_y Miner$ algorithm is evaluated by running it on CoSeLoG Project[6]. The experiments Exp_{NPA}, Exp_{AST} and $Exp_{L\&P}$ was performed on the CoSeLoG Municipality 2, which contains 645 cases and 376 activities. Experiments were conducted and analysed on set of every 100 cases. $\alpha_y Miner$ makes 4 *assumption*: Any activity whose duration is recorded as 0 millisecond is considered as never been executed, since the nanosecond time is not recorded, vocabulary of an activity is not taken into account [1], don't deal with Live or Dead locks and assume that all process have same starting activity.

5.1 Design of Experiment

The $\alpha_y Miner$ experimental set up is shown in Figure [5]. Where the log is first cleaned and initialized using initializer from which the NPA is identified. Optimal resource for performing NPA is identified and their behaviour is analysed. Finally $\alpha_y Miner$ recommends the best *process and resource model.*

[6] http://dx.doi.org/10.4121/uuid:26aba40d-8b2d-435b-b5af-6d4bfbd7a270

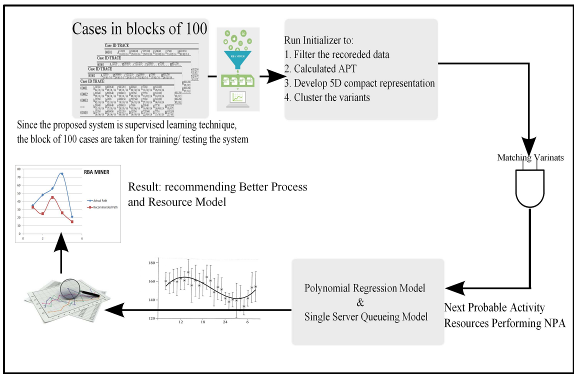

Cases in blocks of 100
Run Initializer to:
1. Filter the recoreded data
2. Calculated APT
3. Develop 5D compact representation
4. Cluster the variants
Since the proposed system is supervised learning technique,
the block of 100 cases are taken for training/ testing the system
Matching Varinats
RBA MINER
Result: recommending Better Process
and Resource Model
Polynomial Regression Model
&
Single Server Queueing Model
Next Probable Activity
Resources Performing NPA

Fig. 5: **Illustration of Online Process Model**

5.2 Recommendation of Next Probable Activity (NPA): Exp_{NPA}

Experiment was simulated in the form of supervised learning, where the test Exp_{NPA} was conducted for every 100 cases and starting from 2^{nd} activity of the sequence. Exp_{NPA} was analysed by comparing it with the actual path of execution. The result of this comparison is shown un Figure [6] and on analysis it is studied that the percentage of error rate (*marked by green line*) in recommendation is lesser in later positions of execution when compared to earlier positions. The Exp_{NPA} achieved *72.8568%* of efficiency. On analysing the graph, it is understood that the behaviour of recommended path is always below the actual path of execution. Inclination shows the huge difference of behaviour between the actual and recommended path. For the cases 400 to 500, it is observed that the graph don't have red line, as the path of execution is critical and was observed to take optimal time for completion. Hence this proves that $\alpha_y Miner$, don't recommend if the path of execution is observed to be optimal.

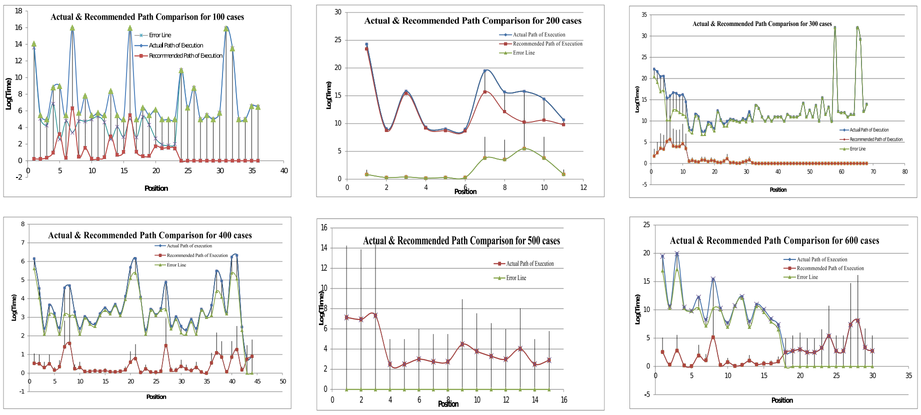


Fig. 6: **Result of Exp**$_{NPA}$

5.3 Recommendation of Resource capable for performing NPA: Exp_{AST}

The Exp_{AST} for each resource performing NPA. Waiting time of recommended resource was compared with the actual resource and it was studied that their performance was improved by *59.7303%*. The Figure [7] show the result of Exp_{AST}. The Exp$_{AST}$, discovered the better path of execution based on resource average service time and it is also understood $\alpha_y Miner$, don't recommend if the resources to whom the task is assigned is efficient in performing.

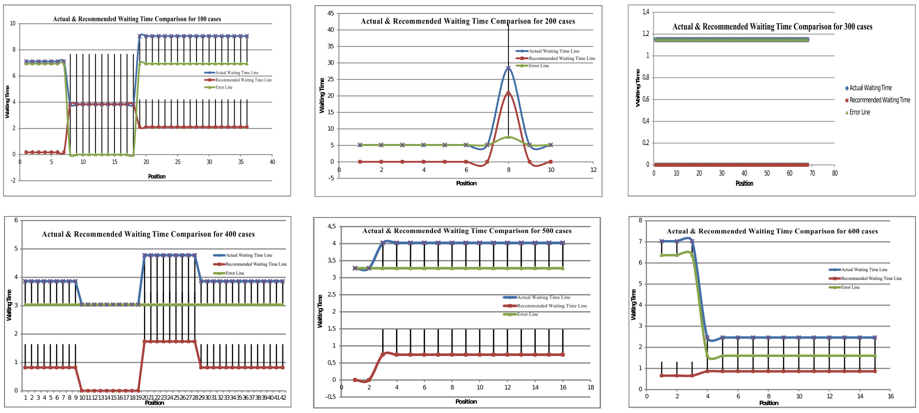


Fig. 7: **Result of Exp$_{AST}$**

	100	200	300	400	500	600	Overall
560530	0.0178571	0.005128	0.005525	0.006329	0.005495	0.009009	0.000787
560598	0.1666667	0.083333	0.166667	0.333333	0.111111	0.090909	0.016949
560521	0.0714286	0.090909	0.083333	0.012987	0.052632	0.008621	0.004587
560532	0.0076336	0.005102	0.009009	0.007194	0.003279	0.005051	0.000517
4634935	0.1428571	0.083333	0.142857	0.043478	0.009709	0.016129	0.006329
560458	0.0069444	0.007519	-0.00115	0.00304	0.00625	0.006369	0.00036
560429	0	1	1	1	1	1	1
560528	0	1	0.5	1	0.5	1	0.166667
560519	0.0153846	0.009009	0.013699	0.01	0.007246	0.001605	0.001754

Table 4: **Result of Average Waiting time for CoSeLoG project**

5.4 Polynomial regression model: $Exp_{L\&P}$

The result of $Exp_{L\&P}$ is shown in Table [5] and the Figure [8] shows the polynomial curve. Using the law of *Arousal*, the optimal load and performance at each resource can be identified. This result is used in making appropriate decision about resource behaviour and load assignments. Using the outcome of experiment proper recommendations can be made, whether to assign the task to that resource ot not.

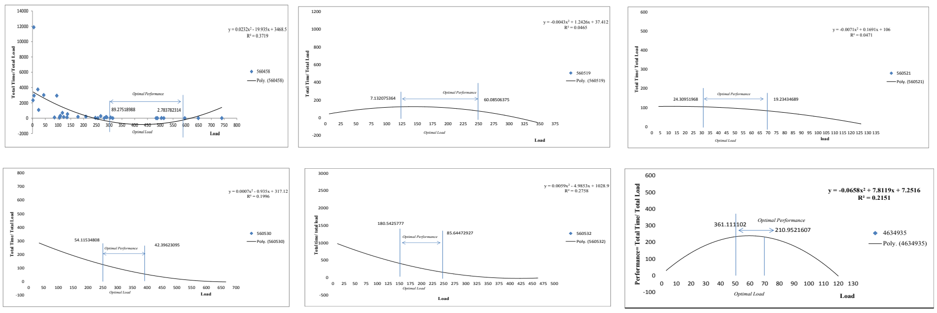


Fig. 8: **Result of** $\boldsymbol{Experiment}_{Load\&Performance}$

Resources	No of load	total time	R^2	Load Range	Performance Range
4634935	744	124351.8	0.2151	50-70	210.9521607 - 361.111102
560458	7838	1161852	0.3719	300-600	2.783782314 - 89.27519
560519	4809	390477.9	0.0465	125-250	7.132075 - 60.08506
560521	1475	92446.43	0.0471	30-70	19.23435 - 24.30952
560530	11140	905091.6	0.1996	250-400	42.39623 - 54.11535
560532	7817	1221671	0.2758	150-250	180.5426 - 85.64473

Table 5: **Result of Polynomial Regression for CoSeLoG project**

6 Conclusion

$\alpha_y Miner$ provided a solution for *recommending* an optimal *path of execution: NPA* along with the *complete process model* and *resource preferable for performing NPA*. $\alpha_y Miner$ is a analytical tool which gave solution for real time business process execution, by analysing the process and resource behaviour. The Experimental result shows 72% of optimization in *process execution* and 59% improvement in the behaviour of resource based on their *Average waiting time, load and performance.* $\alpha_y Miner$ was successful in *recommending appropriate process and resource model* for the real time process.

References

1. Joos CAM Buijs, Boudewijn F van Dongen, and Wil MP van der Aalst. Towards cross-organizational process mining in collections of process models and their executions. In *Business Process Management Workshops*, pages 2–13. Springer, 2012.

2. Justus Klingemann, Jurgen Wasch, and Karl Aberer. Deriving service models in cross-organizational workflows. In *Research Issues on Data Engineering: Information Technology for Virtual Enterprises, 1999. RIDE-VE'99. Proceedings., Ninth International Workshop on*, pages 100–107. IEEE, 1999.
3. Diimitrios Georgakopoulos, Mark Hornick, and Amit Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and parallel Databases*, 3(2):119–153, 1995.
4. Gustavo Alonso, Divyakant Agrawal, Amr El Abbadi, and Carl Mohan. Functionality and limitations of current workflow management systems. *IEEE Expert*, 12(5):105–111, 1997.
5. Asuman Dogac. *Workflow management systems and interoperability.* Number 164. Springer Science & Business Media, 1998.
6. Andrzej Cichocki. *Workflow and process automation: concepts and technology.* Springer Science & Business Media, 1998.
7. Wil Van Der Aalst. *Process mining: discovery, conformance and enhancement of business processes.* Springer Science & Business Media, 2011.
8. J.C.A.M.; Buijs. Environmental permit application process (wabo), coselog project, 2014.
9. Joyce Nakatumba and Wil MP van der Aalst. Analyzing resource behavior using process mining. In *Business Process Management Workshops*, pages 69–80. Springer, 2010.

Capturing the Sudden Concept Drift in Process Mining

Manoj Kumar M V, Likewin Thomas, and Annappa B

Department of Computer Science and Engineering
National Institute of Technology Karnataka, Surathkal
Mangalore - 575025
INDIA
{manojmv,likewinthomas}@nitk.ac.in, annappa@ieee.org
http://www.cse.nitk.ac.in

Abstract. Concept drift is the condition when the process changes during the course of execution. Current methods and analysis techniques existing in process mining are not proficient of analyzing the process which has experienced the *concept drift*. State-of-the-art process mining approaches consider the process as a static entity and assume that process remains same from beginning of its execution period to end.

Emphasis of this paper is to propose the technique for *localizing* concept drift in *control-flow* perspective by making use of *activity correlation strength* feature extracted using process log. Concept drift in the process is localized by applying statistical hypothesis testing methods. The proposed method is verified and validated on few of the real-life and artificial process logs, results obtained are promising in the direction of efficiently localizing the sudden concept drifts in process-log.

Keywords: Concept drift, process mining, event class correlation, activity correlation strength, sudden drift

1 Introduction

Process mining is a fairly new research discipline that stands between process modeling and analysis on the one hand, and computational intelligence and data mining on the other hand. The idea of process mining is to discover, monitor and improve the operational, electronic and embedded processes by using the data logged in process logs[4].

Process mining comprises (automated) process discovery (i.e., mining process models), conformance checking (i.e., monitoring deviances by matching model and log), social network/ organizational mining, automated creation of simulation models, model extension, model repair, case prediction, and history-based recommendations as shown on fig. 1.

There are two main reasons for the increasing attention in process mining. First, more and more events are being logged, thus, providing thorough info about the past of processes. Second, there is a necessity to develop and upkeep business processes in modest and quickly altering environments.

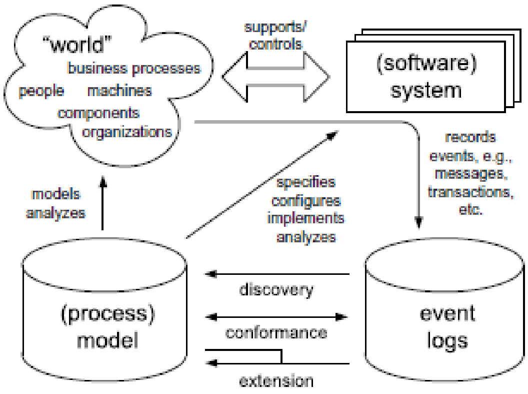


Fig. 1. Basic objectives and types of process mining [**?**]

Process mining techniques offer a means to more rigorously check compliance and ascertain the validity and reliability of information about an organization's core processes.

Beginning point for process mining is availability of appropriate event log. All process mining methods assume that it is possible to sequentially record events. Each event refers to an activity (i.e., a well-defined step in some process) and is related to a particular case (i.e., a process instance). Event logs may store extra info about events. In fact, whenever possible, process mining techniques use extra information such as the resource (i.e., person or device) executing or initiating the activity and time-stamp of the event etc.

Remaining sections of this paper are structured as follows. Section 2 discusses about concept drift with brief and concise example. Section 3 gives the brief description about the terminologies and notations used in this paper. Section 4 briefs about the methodology used to localize the sudden concept drift. Results of our experiments are given in section 5, brief about the related literature is explained in section 6 and this paper ends with some concluding remarks.

2 Concept drift

Process-centric analysis methods and techniques available in process mining are capable of generating excellent insight on working of operational process. If the process is not of static in nature, presently available process mining methods cannot be applied for the analysis. The main erroneous assumption that all of the available process mining techniques does is, *"Process at the end of its execution is same as the process at the beginning of its execution"*[12], this is not often the case due to the possibility of process change during the period of execution. All currently available process mining algorithms fail consider the changes happened in the process during the process execution.

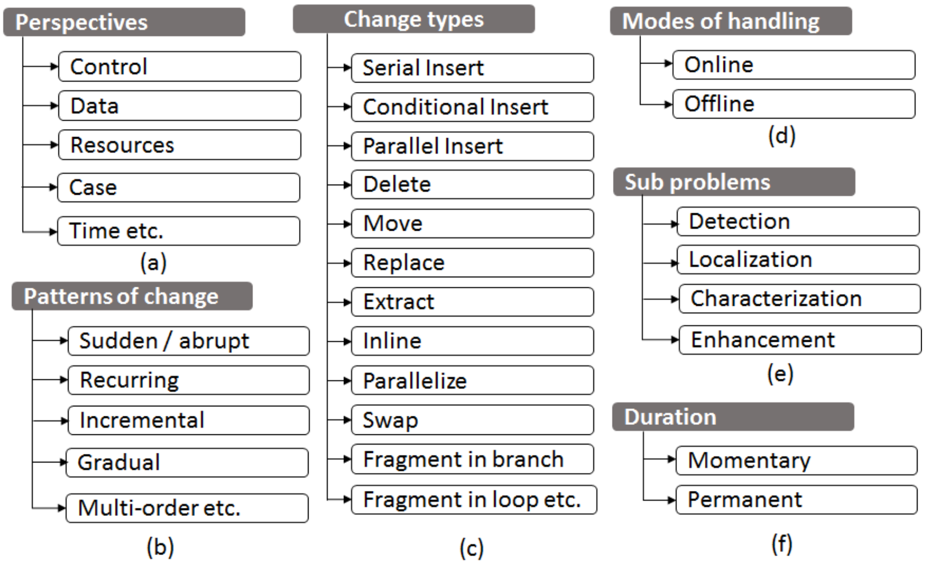


Fig. 2. Concept drift problem dimension

Possibility of occurrence of concept drift has unfortunately been neglected while proposing methods available in the area of process mining. Not concentrating and ignoring the changes in the process makes end results of analysis obsolete.

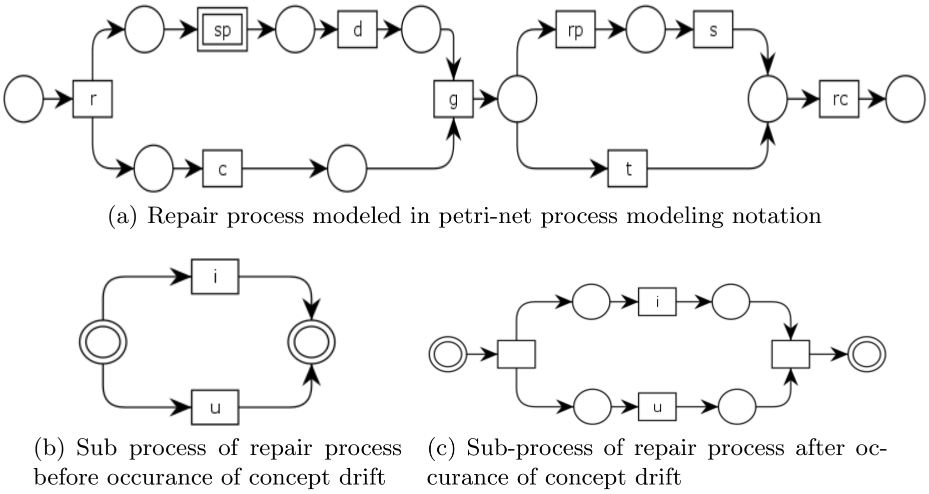
End-to-end Solution for the phenomenon of concept drift can only be achieved by considering *sub-problems involved, perspectives of change, change types, change patterns* and *duration of change* in to account, same is shown in fig. 2 *Change detection* and *change localization* are the two major *sub-problems. Control-flow, data, case* and *organizational* are four the main process perspectives. *Sudden, recurring, incremental* and *gradual* are the four different change types those can be normally observed. Most normally observed change *patterns of change* in control-flow perspective are shown in fig. 2(c). Please refer [7,6,5] to get to know more about different control-flow, resource and data patterns that can be observed in operational process.

For example, consider the process model shown in fig. 3(a) represent the *repair process* of electronic products in a company and is modeled with petri-net notations. A petri net is a bipartite graph consists of places (circle) and transition (rectangle). A transition becomes enable when each of its input places has at least one token in it. Upon firing of transition, it consumes a token from each of its input places and produces a token in each of its output places. The

Table 1. Example process-log

Trace set-1		*Trace set-2*	
t_1	{r, i, c, d, g, rp, s, rc}	t_9	{r, i, u, c, d, g, rp , s, rc}
t_2	{r, u, d, c, g, rp, s, rc}	t_{10}	{r, u, i, c, d, g, rp, s, rc}
t_3	{r, i, c, d, g, t, rc}	t_{11}	{r, i, u, c, d, g, t, rc}
t_4	{r, u, d, c, g, t, rc}	t_{12}	{r, u, i, c, d, g, t, r, c}
t_5	{r, i, d, c, g, rp, s, rc}	t_{13}	{r, i, c, u, d, g, rp, s, rc}
t_6	{r, u, c, d, g, t, rc}	t_{14}	{r, c, i, u, d, g, t, rc}
t_7	{r, i, d, c, g, rp, s, rc}	t_{15}	{r, c, i, u, d, g, rp, s, rc}
t_8	{r, i, d, c, g, t, rc}	t_{16}	{r, i, u, d, c, g, rp, s, rc}

fig. shown in 3 is drawn using Colored Petri-Net[1] Tools (CPNtools[2]). Process model in fig. 3(a) has set of 10 different activities. In fig. 3(a), transition *sp* with double rectangle represents *sub-process*.



(a) Repair process modeled in petri-net process modeling notation

(b) Sub process of repair process before occurance of concept drift

(c) Sub-process of repair process after occurance of concept drift

Fig. 3. Example illustrating *sudden* concept drift in *control-flow perspective* of repair process

Activities of the process in fig. 3(a) are *r=receive repair request, i=inspect item, u=update database, c=check warranty, d=decide the cost of repair, g=get the approval from customer, rp=repair product, s=send bill and collect charges,*

[1] Coloured Petri nets (CPN) are a backward compatible extension of the concept of Petri nets. CPN preserve useful properties of Petri nets and at the same time extend initial formalism to allow the distinction between tokens.

[2] http://www.cpntools.org

t=terminate the repair process and *rc= return item and close case.* Table 1 shows the traces of the *repair process.* According to the process log shown in table 1, process experiences *concept drift* after t_8 i.e. the traces t_1 to t_8 represents the process traces before change and t_9 to t_{16} are the traces possible after process change.

Before concept drift (before t_9), any one of the activities *inspect item* or *update database* can be observed in traces of the log shown in table 1. After the occurrence of concept drift (after t_8), both *inspect item* and *update database* activities can be observed. This example precisely signify the effect of concept drift in process. If we employ the process discovery methods available in process mining to construct the process model using the process log shown in Table 1, outcome will be process model in the fig. 3 with the excerpt shown in fig. 3(a) as the subprocess replacing the activity *sp*.

3 Event class and Event class correlation

Let $\mathcal{A}$ be a set of activity names. A trace σ is a sequence of activities, i.e., $\sigma \in \mathcal{A}^*$. A simple event log L is a multi-set of traces over $\mathcal{A}$, i.e., $L \in \mathbb{B}(\mathcal{A}^*)$

Definition (Event, log trace, log). Let E be a set of unique set of log events. l is a log trace over E if and only if l is a non-repeating sequence on E . A set of log traces L is a log over E if and only if all log traces $l \in L$ are log traces over E and $\forall_{l_1, l_2} \in L : (set(l_1) \cap set(l_2) \neq \emptyset) \rightarrow (l_1 = l_2)$.

Using the definition of event, trace and log, event class can be defined as follows.

Definition (Event class). $c \in E \rightarrow C$ maps each event to its event class, where C is the set of event classes.

The set of event classes for a log trace l can be defined as follows:

$$C(l) = \{c(e) | e \in l\}$$

The set of event classes for a log L is defined as follows:

$$C(L) \cup_{l \in L} C(l)$$

Let C be a set of event classes. The function $ecc \in C \times C \rightarrow \mathbb{R}_0^+$ assigns to each tuple of event classes a certain correlation value. The larger this the value is, the more related the two respective event classes are.

In our method we define the correlation function among event classes by scanning the whole log. We begin with a matrix of $C \times C$, set with zero values before the real scanning pass. While traversing the log, this matrix is updated for every following relation that is found. Correlation matrix, as well as the correlation function itself, is symmetric, i.e., $ecc(X, Y) = ecc(Y, X)$. During the scanning pass, this regularity requires to be preserved by the algorithm.

Consider the fig. 4, the scanning is presently examining an event of class e_1. We call the event presently under consideration as *reference event.* Looking at the directly preceding event of class e_2, the scanner can establish an observation

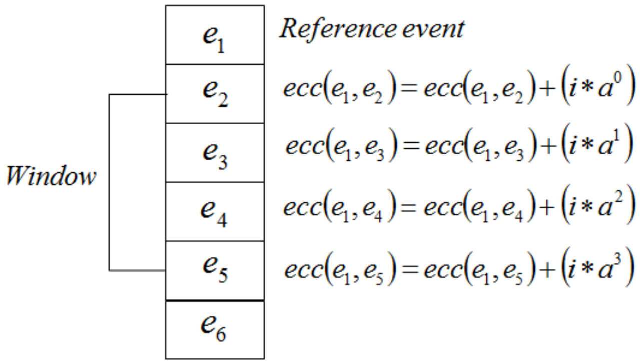

e_1
Reference event
e_2
$ecc(e_1,e_2) = ecc(e_1,e_2) + (i * a^0)$
e_3
$ecc(e_1,e_3) = ecc(e_1,e_3) + (i * a^1)$
Window
e_4
$ecc(e_1,e_4) = ecc(e_1,e_4) + (i * a^2)$
e_5
$ecc(e_1,e_5) = ecc(e_1,e_5) + (i * a^3)$
e_6

Fig. 4. *ecc* calculation

of the co-occurrence between event classes e_1 and e_2, which means that their association is strengthened. Similarly, the correlation matrix value for $ecc(e_1, e_2)$ is incremented by i, the increment value (generally set to 1). In our method, the scanning pass uses a look-forward window for calculating each event. This means that if the look-forward windows size is seven, the scanner will consider the upcoming seven events which have followed the reference event. When calculating events in the look forward window, the scanner will weaken its measurement exponentially, based on an attenuation factor a, where $0 \leq a \leq 1$.

For any event y in the look-forward window, where x is the reference event, the correlation matrix will be updated as given below

$$ecc(c(x), c(y)) = ecc(c(x), c(y)) + (i.a^n) \tag{1}$$

where n is the number of events located between x and y in the trace.

After the scanning pass has estimated all events in all traces of the log, a trustworthy correlation function between event classes is recognized, as expressed in the aggregated correlation matrix. Our correlation function thus relates two event classes as more linked, if events of these classes commonly happen closely together in traces of the log.

Concept drift is the condition where the process experiences change during the course of analysis. We believe that the representative appearance of feature values change before and after the occurrence of concept drift. By considering the sequential order of process instances in the log, we apply *windowing* strategy for selecting the instances for processing and to localize the occurrence of concept drift. *Statistical hypothesis tests*[3] are used to examine differences between successive feature values obtained using *event class correlation.*

4 Methodology

[3] Hypothesis testing is really a systematic way to test claims or ideas about a group or population, using data measured in a sample.

Algorithm 1 Algorithm to detect concept drift using event class correlation

```
Require: Process log with concept drifts
$sub\_logs \leftarrow 0$ // set the initial value to 0
$sub\_logs \leftarrow split\_log(process\_log, size)$
$num\_sub\_logs \leftarrow sub\_logs.size()$
while $num\_sub\_logs \neq 0$ do
    $i \leftarrow 0$
    $activities = get\_activities\_of\_sub\_log(sub\_log[i])$ // get the number of activities in the each sub log
    $i \leftarrow i + 1$
    $cor[size(activities)][size(activities)] \leftarrow 0$
    for $\forall case_i \in sub\_logs_i$ do
        $sub_case \leftarrow 0$
        for $\forall event_i \in case_i$ do
            $look\_back \leftarrow 0$ 1
            for $\forall events_j \in case_i$ do
                if name$(event_i \neq event_j)$ then
                    if $look\_back \leq size$ then
                        $cor[event_i][event_j] \leftarrow cor[event_i][event_j]+(i \times a^{look\_back})$ // calculate the ecc of event classes $i$ and $j$
                        $look\_back = look\_back + 1$
                    end if
                end if
            end for
        end for
    end for
    $num\_sub\_logs \leftarrow num\_sub\_logs - 1$
    $level\_of\_significance = 0.05$ // Set the level of significance (alpha value)
    $Test\_satistic = test\_hypothesis(cor, hypothesis\_test\_name, window\_size, num\_of\_popultions)$ // (performing hypothesis tests)
    $P\_value \leftarrow Compute\_P - value(Test\_statistic)$
    if $P\_value \leq level\_of\_significance$ then
        Reject $\mathcal{H}_0$ and declare concept drift // deciding the validity of $\mathcal{H}_0$
    end if
end while
```

The standard process of statistical hypothesis testing comprises of four phases

- ***S***$_1$***:*** Formulating *null* ($\mathcal{H}_0$) and *alternative* hypothesis ($\mathcal{H}_1$)
- ***S***$_2$ ***:*** Identifying a *test statistic* that can be used to assess the trustworthiness $\mathcal{H}_0$.
- ***S***$_3$ ***:*** Calculate the P-value (probability of obtaining a sample outcome, given that the $\mathcal{H}_0$ is true).
- ***S***$_4$ ***:*** Compare the P-value to a statistical significance level α. If $P \leq \alpha$, that the observed effect is statistically significant, $\mathcal{H}_0$ is ignored, and the $\mathcal{H}_1$ is considered as valid.

$\mathcal{H}_0$ can be stated as,

($\mathcal{H}_0$): There is no significant characteristic differences in the manifestation of consecutive populations of *feature* values.

Null hypothesis is considered as fact until proved as false. When the null hypothesis is proved as false, alternative hypothesis ($\mathcal{H}_1$: There is significant difference in manifestation of feature values) is considered and accepted and occurrence concept drift is declared.

Complete procedure for assessing the hypothesis tests on consecutive populations of *ecc* values is shown in the algorithm 1. We choose *two-sample* (since we need to analyze two samples of the population at the given point of time for detecting concept drift), *independent* (since both the samples are not depending on each other), *non-parametric*(since we do not know the priori distribution of the feature values in an event log), *uni-variate* and *multi-variate* (univariate tests deal with scalar data and multivariate tests deal with vector data) statistical hypothesis tests for detecting and localizing the concept drift in the process.

Using windowing strategy as instance selection method, successive populations of feature values are compared and examined to discover any significant difference. Significant difference between feature values only observed during the change in the process. Depending on the requirement of our problem and based on the characteristics of the tests described in the previous paragraph we consider *Mann-Whitney U Test* and *The Moses Test for Equal Variability. Mann-Whitney U Test* is used to answer ***"do two independent samples represent two populations with different median values"*** (or different distributions with respect to the rank-orderings of the scores in the two underlying population distributions)? *The Moses Test for Equal Variability* test will be used to answer ***Do two independent samples represent two populations with different variances?***

5 Experiments and Results

Table 2. Process-log details and concept drift locations

	Process log	**Cases**	**Activities**	**Events**	$\mathbf{c}_{reo}$	$\mathbf{c}_{rep}$	$\mathbf{c}_{ins}$	$\mathbf{c}_{del}$
L_1	Loan application process	13,087	36	2,62,200	5,000	7,500	-	-
L_2	Volvo IT incident management process	7,554	13	65,533	-	3,000	4,000	-
L_3	Insurance claim process	500	21	7,033	-	-	200	400

Process before the occurrence of concept drift represent different version of the process than after the occurrence of concept drift. Concept drift can be observed in the process any number of times.

It is very hard to find real-life operational process-log with concept drift in it. Process mining doesn't has any standard data set or workbench for testing the credibility of algorithms detecting and localizing concept drift. There are

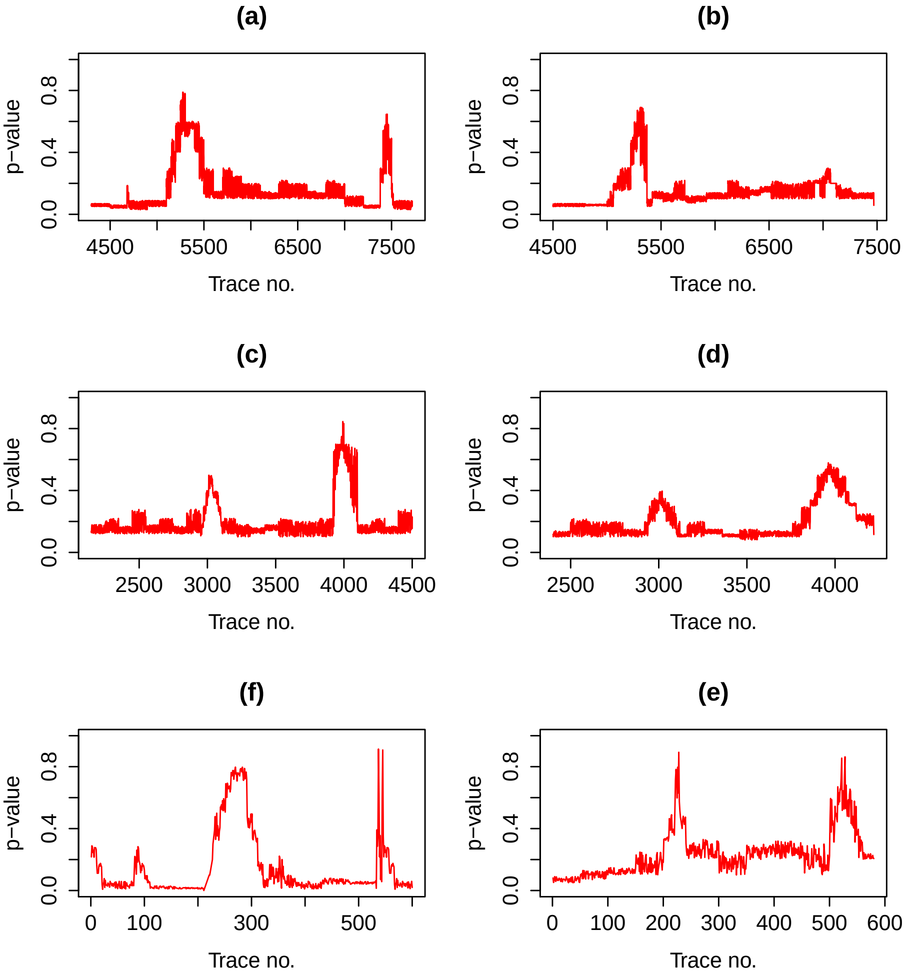


Fig. 5. Result of application of statistical hypothesis testing on process logs L_1, L_2 and L_3 given in 2. Graphs (a),(c),(e) are the results of application of *Mann-Whitney U Test* and the graphs (b), (d), (f) are the results of application of *The Moses Test for Equal Variability* on process logs L_1, L_2 and L_3 respectively

few real-life standard datasets available[3] [4], but they are not appropriate for testing the algorithms dealing with concept drift. In our experiments, we have taken appropriate data sets from open repository of process logs and artificially induced concept drift in the control-flow perspective of the process.

[3] http://data.3tu.nl/repository/

[4] http://www.processmining.org/logs/start

Process logs form the open process log repository are used and modified to include concept drift. We used Colored Petri Net (CPN) Tools with CPNXES library[5] for creating synthetic process logs. Approach proposed in this paper is tested on 3 different logs shown in table 2.

- ***c_{reo}:*** Rearranging activities.
- ***c_{rep}:*** Replacing one activity with other.
- ***c_{ins}:*** Inserting a new activity.
- ***c_{del}:*** Deleting an existing activity.

Table 2 lists info about three different process logs that we have considered to validate the approaches proposed in this paper. Process logs L_1 (*Loan application process*) and L_2(*Volvo IT incident management process*) are taken from on-line process log repository. Process log L_3 (*Insurance claim process*) is generated with the help of CPNTools and CPNXES library. Both L_1 and L_2 are modified to include concept drift[1]. Table 2 shows the details about location of concept drift in L_1, L_2 and L_3.

Process log L_1 is altered to include the concept drift caused by sudden c$_{reo}$ and c$_{rep}$. L_2 is altered to include sudden c$_{rep}$ and c$_{ins}$. Process log L_3 is altered to include sudden c$_{ins}$ and c$_{del}$. By considering the limitation of space, we have ignored to include the models of the process L_1, L_2 and L_3 in this paper.

$\mathcal{H}_0$ is assessed on each of L_1, L_2 and L_3 by applying hypothesis tests discussed in the last section, result is shown in fig. 5. Graphs a,c,f in fig. 5 are the results of application of *Mann-Whitney U Test* and graphs b,d,e in fig. 5 are the results of application of *The Moses Test for Equal Variability* on process logs described in the table 2. Crests in the graphs given in fig. 5 signify real concept drift in the process. There are some false alarms are also be possible, one should be very careful about the actual concept drift and the concept drift caused by noise in the process log. The drastic change in the P-value is observed during the occurrence of concept drift. Our technique perform poorly when applied to localize the concept drift caused by c$_{rep}$ and same can be seen in the graphs shown in fig. 5 (b) and (c) (at the trace number 7500 in 5 (b) and 3000 in 5 (c)). Overall, results shown in graph 5 implies that the methods presented in this paper for localizing concept drift is promising. In future, the proposed technique is going to be added to ProM, and there by making it possible for people to experiment with it.

6 Related work

The word *concept drift* is initially coined by Schlimmer et.al. during 1986 in the article *Incremental learning from noisy data*[8]. Phenomenon of concept drift is known by many terminologies in other research disciplines (as Covariate Shift

[5] https://westergaard.eu/2011/07/prom-package-documentation-keyvalue/

[1] process logs and models used in this paper can be downloaded at http://www.cse.nitk.ac.in/researchscholars/manoj-kumar-m-v

in machine learning, as Load Shedding in databases, as Temporal Evolution in Information retrial etc.). Efficiently handling concept drift is an important concern in every data analysis disciplines[2], unfortunately it has been deeply neglected in process mining. According to [2], concept drift is a *non stationary learning problem over time* and [1] describes drift as the *process of changing the process*. The core theory when dealing with the concept drift problem is *uncertainty* about the future. It can be *assumed, estimated or predicted but there is no certainty.*

Some efforts have been made to find different versions of control-flow perspective of the process using clustering and classification techniques available in Data Mining[9,10,11]. Finding different versions of the process does not consider the *type, pattern* and *perspective* of concept drift. Hence, they cannot be the suitable means for solving phenomenon of concept drift.

ProM is the open source process mining framework consisiting more than $1,200$[6] plug-in and plug-in variants that can be used for solving different process mining problems, out of which one or two plug-ins capable of addressing the problem of concept drift. To our knowledge, two works in the literature that addresses concept drift in process mining are [12,14,13]. Technique proposed in [12] are tested on real setting and the results are documented in [13]. Both [12,13] proposes extracting different global and local features out of process log and applying statistical hypothesis testing for detecting and localizing concept drift. Techniques shown in [12,14] propose solution for *offline* and *online* methods for detecting and localizing *sudden* concept drift in *control-flow* perspective of process. The idea of extracting Event Class Correlation (ecc) feature is taken form [3]. End-to-end solution for the problem of concept drift can only be accomplished if it is addressed by considering all *perspectives, types and patterns* of change shown in fig.2. Effort given in this paper suggests the method of localizing sudden concept drift in the control-flow perspective of the process using event class correlation feature by applying statistical hypothesis testing methods.

7 Conclusion

Handling the phenomenon of concept drift efficiently is the prime concern in all disciplines that deal with data analysis. Concept drift is the situation when process experiences changes in its associated perspectives during the period of its execution. The configuration of the process before the occurrence of concept drift is different from the process after the occurrence of concept drift. State-of-the-art process-centric analysis techniques available in process mining behave poorly when employed to analyze the process that has experienced concept drift. Because, they consider the process as a static entity. But, process represents the dynamic aspect of the organization and can evolve in any perspective showing any change pattern exhibiting several different change type during the phase of its execution. This paper proposes the extraction of *event class correlation*

[6] http://www.promtools.org/doku.php?id=packdocs

feature for localizing the sudden concept drift in the control-flow perspective of the operational process. Results of the experimental study shown that proposed methods are capable of localizing concept drift efficiently. Our feature work include extension of the proposed methods to make working in on-line setting for sudden and gradual drift detection and localization.

References

1. Gama, Joo, et al. *"A survey on concept drift adaptation."* ACM Computing Surveys (CSUR) 46.4 (2014): 44.
2. Zliobaite, Indre. *Learning under concept drift: an overview. Overview*, Technical report, Vilnius University, 2009 techniques, related areas, applications Subjects: Artificial Intelligence, 2009.
3. Gnther, Christian W., Anne Rozinat, and Wil MP Van Der Aalst. *"Activity mining by global trace segmentation."* Business process management workshops. Springer Berlin Heidelberg, 2010.
4. Van Der Aalst, Wil, et al. "Process mining manifesto." Business process management workshops. Springer Berlin Heidelberg, 2012.
5. Russell, Nick, Ter Hofstede, Arthur HM, Mulyar, Nataliya, *Workflow controlflow patterns: A revised view*, Citeseer, 2006.
6. Ter Hofstede, Arthur HM, David Edmond, and Wil MP van der Aalst. *"Workflow resource patterns"* (2005): 13-17.
7. Russell, Nick, Arthur HM Ter Hofstede, David Edmond, and Wil MP van der Aalst. *Workflow data patterns.* QUT Technical report, FIT-TR-2004-01, Queensland University of Technology, Brisbane, 2004.
8. Schlimmer, Jeffrey C., and Richard H. Granger Jr. *"Incremental learning from noisy data."* Machine learning 1.3 (1986): 317-354.
9. Song, Minseok, Christian W. Gnther, and Wil MP Van der Aalst. *"Trace clustering in process mining."* Business Process Management Workshops. Springer Berlin Heidelberg, 2009.
10. Luengo, Daniela, and Marcos Seplveda. *"Applying clustering in process mining to find different versions of a business process that changes over time."* Business Process Management Workshops. Springer Berlin Heidelberg, 2012.
11. Bose, RP Jagadeesh Chandra, and Wil MP van der Aalst. *"Context Aware Trace Clustering: Towards Improving Process Mining Results."* SDM. 2009.
12. Bose, RP Jagadeesh Chandra, et al. *"Handling concept drift in process mining."* Advanced Information Systems Engineering. Springer Berlin Heidelberg, 2011.
13. Bose, RP Jagadeesh Chandra, et al. *"Dealing with concept drifts in process mining."* Neural Networks and Learning Systems, IEEE Transactions on 25.1 (2014): 154-171.
14. Carmona, Josep, and Ricard Gavalda. *"Online techniques for dealing with concept drift in process mining."* Advances in Intelligent Data Analysis XI. Springer Berlin Heidelberg, 2012. 90-102.