Chair of Process and Data Science

RWTH AACHEN UNIVERSITY

Model Repair by Incorporating Negative Instances In Process Enhancement

Master Thesis

Author : **Kefang Ding**

Supervisor : Dr. Sebastiaan J. van Zelst

Examiners : Prof. Wil M.P. van der Aalst
Prof. Thomas Rose

Registration date : 2018-11-15

Submission date : 2019-04-08

This work is submitted to the institute

PADS RWTH University

Acknowledgments

The acknowledgments and the people to thank go here, don't forget to include your project advice.

Abstract

Process mining is based on business execution history in the form of event log, aim to bring visual insights on the business process and to support process analysis and enhancements. It bridges the gap between traditional business process management and advanced data analysis techniques like data mining and gains more interests and application in recent years.

Process enhancement, as one of the main focuses in process mining, improves the existing processes according to actual business execution in the form of event logs. The records in an event log can be classified as positive and negative according to predefined Key Performance Indicators, e.g. the throughput time, and cost. Most of the current enhancement techniques only consider positive instances from an event log to improve the model, while the value hidden in negative instances is simply neglected.

This thesis provides a novel strategy that considers not only the positive instances and the existing model but also incorporate negative information to enhance a business process. Those factors are balanced on directly-follows relations of activities and generate a process model. Subsequently, long-term dependencies of activities are detected and added to the model, in order to block negative instances and obtain a higher precision.

We validate the ability of our methods to incorporate negative information with synthetic data at first. Then, we conduct experiments in a scientific workflow platform KNIME to show the statistical performance of our methods. The results showed that our method is able to overcome the shortcomings of the current repair techniques in some situations and repair models with a higher precision.

Chapter 1

Preliminaries

This chapter introduces the most important concepts and notations that are used in this thesis. Firstly, the event data and process models typically used in process mining are described. Later, details of Inductive Miner techniques which relate to our work are listed.

1.1 Event Log

Business processes in organizations can be reflected by their activities execution. The historical execution data is usually stored as event logs in information systems and can be used by process mining techniques to analyze, understand, and improve the business execution. To specify the event log, we begin with formalizing the various notations[1] .

Definition 1.1 (Event)**.** An event corresponds to one execution of an activity in business execution and written as e. An event is characterized by attributes, like a timestamp, activity name, associated costs, etc. The set of all possible events in a process is written as $\mathcal{E}$.

Definition 1.2 (Trace)**.** A trace is a finite sequence of events $\sigma \in \mathcal{E}^*$ with conditions that (i) each event appears only once in a trace.

$$\forall 1 \leq i, j \leq |\sigma|, i \neq j \Rightarrow \sigma(i) \neq \sigma(j)$$

(ii) one event can only appear in one trace.

$$\forall e \in \sigma, e \in \sigma\prime \Rightarrow \sigma = \sigma\prime$$

A trace also has a set of attributes, like its unique identifier, the cost. We extend this definition to handle traces with performance output according to certain KPIs.

Definition 1.3 (Labeled Trace)**.** A trace is labeled with respect to certain KPIs, if it has an attribute for the performance output. We call a trace positive, if the value of its performance attribute is positive, else the trace is negative.

Definition 1.4 (Event log and labeled Event log)**.** An event log L is a set of traces, $L \in \mathcal{B}(\mathcal{E}^*)$. A labeled event log is an event log if all of its traces have an performance attribute according to certain KPIs.

1.2 Process Models

After gathering an event log from the information system, process mining can discover a process model based on the event log, aims to improve the understanding of the business process. To describe the process, multiple process modeling languages are proposed in the last years, e.g, Petri net, BPMN models, etc.

Among those model languages, Petri net has been best studied thoroughly. It captures concurrent systems in a compact manner. Process trees are based on a tree structure to organize the event relation and simple to understand in comparison with other models, like BPMN models. In this thesis, we use Petri net and process tree to represent our process.

1.2.1 Petri Nets

Petri nets are bipartite graphs which are built by ***transitions*** represented by a square and ***place*** in a circle. ***Tokens*** in black dots are put in places to express the states of a Petri net. In the following, we define Petri nets in a formal way.

Definition 1.5 (Petri net)**.** A Petri net is a tuple $N = (P, T, F)$ where
P and T are disjoint finite set of places and transitions, respectively, $P \cap T = \emptyset$.
F is the set of arcs to connect places and transitions, $F \subseteq (P \times T) \cup (T \times P)$.

Further, we can define a labeling function λ on the transitions.

Definition 1.6 (Labeling function λ)**.** λ is a function defined as

$$\lambda : T \to \Sigma \cup \{\tau\}, \Sigma \text{ is a set of labels}, \tau \text{is an empty label.}$$

To express the dynamic states of Petri nets, we introduce a concept called ***marking***.

Definition 1.7 (Marking)**.** A marking is a place multiset,$M : P \to N$, which assigns to a place a number of tokens.

With marking concepts, we extend the form $N = (P, T, F)$ and define a marked Petri net.

Definition 1.8 (Marked Petri nets)**.** A marked Petri net is a 4-tuple $N = (P, T, F, M_0)$ where M_0 is an initial marking of this net.

If all the input places for a transition hold a token, the transition becomes enabled. In other words, the activity in the corresponding process can be triggered. After this execution, the token in the input places are consumed and new tokens are generated in the output places. The firing sequence of transitions can carry out business execution in reality. However, to perform the business on an enterprise level, the correctness of business process models is necessary. Soundness defines a minimum correctness criterion that a process model should satisfy[2]. In the following, we give the definition of soundness for Petri nets.

Definition 1.9 (Soundness)**.** A Petri net is sound if and only if it satisfies the following conditions.

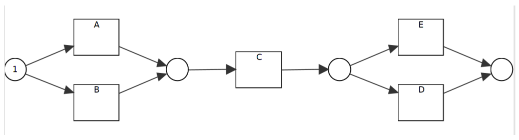
- safeness. Places cannot hold multiple tokens at the same time.

- proper completion. If the sink place is marked, all other places are empty.
- option to complete. It is always possible to reach the final marking just for the sink place.
- no dead part. For any transition there is a path from source to sink place through it.

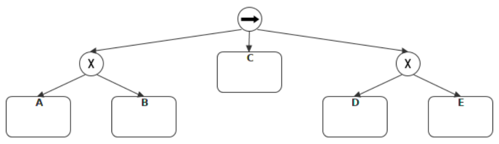
With guarantee of correctness, a subclass of Petri nets is extracted to model the workflow of process activities in real life. This subclass is called workflow nets.

Definition 1.10 (Workflow nets)**.** A workflow net is a Petri net with constraints:
(1) only one source place and one sink place. (2) sound

An example is shown in Figure 1.1a. It has transitions $T = \{A, B, C, D, E\}$ and four places with the initial marking in the place before $T = \{A, B\}$.



(a) A Simple Petri net



(b) A process tree corresponding to Fig 1.1a

1.2.2 Process Tree

Process tree is block-structured and sound by construction, while Petri nets, BPMN models possibly suffer from deadlocks, other anomalies[1]. Here we give the definition of process tree.

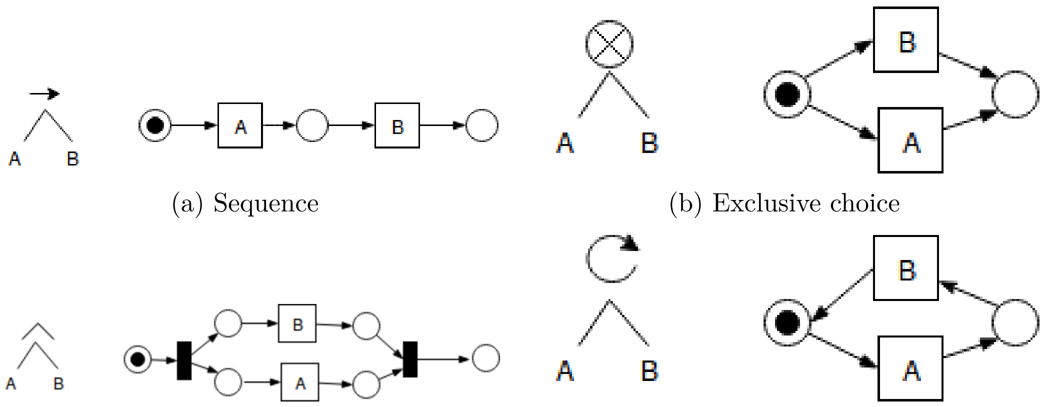
Definition 1.11 (Process Tree)**.** Let $A \subseteq \mathbb{A}$ be a finite set of activities with silent transition $\tau \in \mathbb{A}$, $\bigoplus \subseteq \{\rightarrow, \times, \wedge, \circlearrowleft\}$ be the set of process tree operators.

- $Q = a$ is a process tree with $a \in A$, and
- $Q = \oplus(Q_1, Q_2, ..Q_n)$ is a process tree with $\oplus \in \bigoplus$, and Q_i is a process tree, $i \in 1, 2, .., n, n \in \mathbb{N}$.

Process tree operators represents different block relation of each subtree. Their semantics are standardized from [3, 4] and explained with use of Petri net in Figure 1.2[4].

Definition 1.12 (Operator Semantics)**.** The semantics of operators $\bigoplus \subseteq \{\rightarrow, \times, \wedge, \circlearrowleft\}$ are,

- if $Q =\rightarrow (Q_1, Q_2, ..Q_n)$, the subtrees have sequential relation and are executed in order of $Q_1, Q_2, ..Q_n$
- if $Q = \times(Q_1, Q_2, ..Q_n)$, the subtrees have exclusive choice relation and only one subtree of $Q_1, Q_2, ..Q_n$ can be executed.
- if $Q = \wedge(Q_1, Q_2, ..Q_n)$, the subtrees have parallel relation and $Q_1, Q_2, ..Q_n$ they can be executed in parallel.



(a) Sequence (b) Exclusive choice

(c) Parallelism (d) Loop

Figure 1.2: Semantics of process tree operators w.r.t. Petri net

- if $Q = \circlearrowright (Q_1, Q_2, ..Q_n)$, the subtrees have loop relation and $Q_1, Q_2, ..Q_n$ *with* $n \geq 2$, Q_1 is the do-part and is executed at least once, $Q_2, ..Q_n$ are redo part and have exclusive relation.

According to the corresponding semantic relations, a process tree can be easily transformed into Petri net. In Figure 1.1b, it is the process model in process tree which describes the same process as in Figure 1.1a.

1.3 Inductive Miner

To discover a process model from an event log, we choose one of the leading process discovery approaches – Inductive Miner, because it guarantees the construction of a sound model, and is flexible and scalable to event log data. Its steps are listed bellow.

1.3.1 Construct a directly-follows graph

A the start, the event log L is scanned to extract the directly follows relation of events. The directly-follows relation is like the one in α-algorithm [5, 6], but the frequency information is stored for each relation. Later, those relations are combined together to build a directly-follows graph with frequency. According to [1, 6], a directly-follows graph is defined bellow.

Definition 1.13 (Directly-follows Graph)**.** The directly-follows relation $a > b$ is satisfied iff there is a trace $\sigma\ where, \sigma(i) = a\ and\ \sigma(i+1) = b.$ A directly-follows graph of an event log L is $G(L) = (A, F, A_{start}, A_{end})$ where A is the set of activities in L, $F = (a, b) \in A \times A | a >_L b$ is the directly-follows relation, A_{start}, A_{end} are the set of start and end activities respectively.

The frequency information of the directly-follows relation is called cardinality and defined below.

Definition 1.14 (Cardinality in a directly-follows graph)**.** Given a directly-follows graph G(L) derived from an event log L, the cardinality of each directly-follows relation in G(L) is :

- $Cardinality(E(A, B))$ is the frequency of traces with $\langle ..., A, B, ... \rangle$.

- Start node A cardinality $Cardinality(Start(A))$ is the frequency of traces with begin node A.
- End node B cardinality $Cardinality(End(A))$ is the frequency of traces with end node B.

1.3.2 Split Log Into Sublogs

Based on the directly-follows graph, it finds the most prominent cut which is applied afterwards to split the event log into smaller sublogs. Cuts compose of *exclusive-choice cut, sequence cut, parallel cut and redo-loop cut* which correspond to the process tree operators $\{\rightarrow, \times, \wedge, \circlearrowleft\}$. They are selected in the following order. A maximal exclusive-choice cut is firstly tried to split the directly-follows graph; if it is not available, then a maximal sequence cut, a maximal parallel cut and a redo-loop cut are applied in sequence. Sublogs are created due to this available operator. Meanwhile, this operator is used to build the process tree.

The same procedure is applied again on the sublogs until single activities. What's more, this process tree can be converted into Petri net for further analysis.

Chapter 2

Conclusion

Bibliography

[1] Wil Van der Aalst. Data science in action. In *Process Mining*, pages 3–23. Springer, 2016.

[2] BF Van Dongen, Jan Mendling, and WMP Van Der Aalst. Structural patterns for soundness of business process models. In *2006 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06)*, pages 116–128. IEEE, 2006.

[3] Wil van der Aalst. *Process Mining: Data Science in Action*. Springer Publishing Company, Incorporated, 2nd edition, 2016. ISBN 3662498502, 9783662498507.

[4] Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. On the role of fitness, precision, generalization and simplicity in process discovery. In *OTM Conferences*, 2012.

[5] Wil Van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.

[6] Sander JJ Leemans, Dirk Fahland, and Wil MP van der Aalst. Discovering block-structured process models from event logs-a constructive approach. In *International conference on applications and theory of Petri nets and concurrency*, pages 311–329. Springer, 2013.