RWTH AACHEN UNIVERSITY

Model Repair by Incorporating Negative Instances In Process Enhancement

Master Thesis

Author : **Kefang Ding**

Supervisor : Prof. Wil M.P. van der Aalst
Prof. Thomas Rose
Dr. Sebastiaan J. van Zelst

Registration date : 2018-11-15

Submission date : 2019-??-??

This work is submitted to the institute

PADS RWTH University

Chapter 1

Preliminary

This chapter introduces the most important ground concepts and notations that are used in the remainder of this article. Firstly, the data and process models in process mining are described; Later, one related discovery technique is introduced.

1.1 Event Log

Business process in organizations is reflected by its execution of a set of activities. The historical execution data is usually in a form of event logs in information systems and can be used by Process Mining to analyze the business execution. To specify the event log, we begin with formalizing the various notations[5] .

Definition 1.1 (Event)**.** An event corresponds to an activity in business execution and can be marked by e. An event is characterized by attributes, like a timestamp, name, and associated costs, etc. The finite set of events in the 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 can not appear twice in a trace, $\forall i, j, 1 \leq i, j \leq |\sigma|, if\ i \neq j, then\ \sigma(i) \neq \sigma(j)$. *(ii)* one event can only appear in one trace, $\forall e \in \sigma, if\ e \in \sigma\prime, then\ \sigma = \sigma\prime$. A trace also has a set of attributes, which describes the trace characters, like the identifier, the trace cost.

Definition 1.3 (Event Log)**.** An event log L is a set of traces. If an event log contains timestamps, then the ordering in a trace should respect these timestamps.

1.2 Process Models

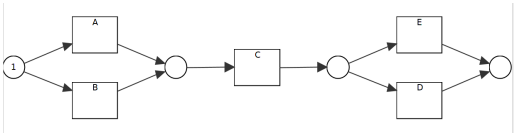
After gathering the event log from information systems, process mining can discover a process model based on the event log, aims to improve understanding and insights on the business process. Multiple process modeling languages are applied in process mining in the last years, such as Petri net, BPMN models, etc. Petri nets are bipartite graph to describe concurrent systems. BPMN models, fully named Business Process Model and Notation models, include many different elements and are flexible but complex tool to visualize business process. Process tree is based on a tree structure to organize the event relation. In this article, due to simplicity, we focus on Petri net and process tree.

1.2.1 Petri Net

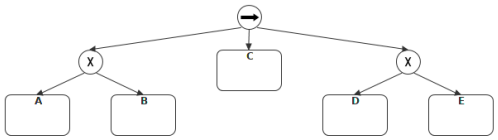
Among multiple models which are proposed to describe the process, Petri net has been best studied to allow for the modeling of concurrency.

Definition 1.4 (Petri net)**.** A Petri net N is composed of a finite set of places P, transitions T, and a set of directed arcs $F \subseteq (P \times T) \cup (T \times P)$, which can be written as $N = (P, T, F)$. A marked Petri net is (P,T,F,M) where M the marking of the net. A marking of a net N is a multi-set over P, $M \in \mathbb{P}$.

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[5]. Here we give the definition of process tree.

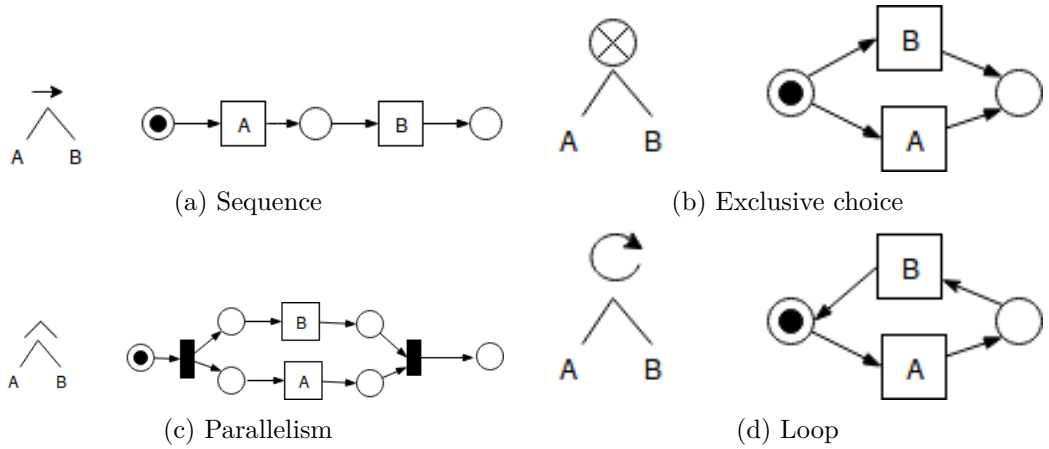
Definition 1.5 (Process Tree)**.** Let $A \subseteq \mathbb{A}$ be a finite set of activities with silent transition $\tau \in \mathbb{A}$, $\bigoplus \subseteq \{\rightarrow, \times, \wedge, \circlearrowleft\}$ be the set of process tree operators.

- $Q = a$ 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 [6, 1] and explained with use of Petri net in Figure 1.2[1].

Definition 1.6 (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.
- if $Q = \circlearrowleft (Q_1, Q_2, ..Q_n)$, 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.



(a) Sequence (b) Exclusive choice

(c) Parallelism (d) Loop

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

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 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 [7, 3], 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 [5, 3], a directly-follows graph is defined bellow.

Definition 1.7 (Directly-follows Graph)**.** The directly-follows relation $a > b$ is satisfied iff there is a trace σ *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.

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

Implementation

ProM is an open-source extensible framework. It supports wide process mining techniques in the form of plug-ins[4]. The algorithm to incorporate negative information is implemented as one plug-in called ***Repair Model By Kefang*** in ProM and released online[2]. This chapter is divided into four parts to describe the whole implementation. Firstly, the inputs and outputs of this implementation is introduced. After accepting the inputs, a directly-follows graph is constructed by dfg-method and later converted into process tree or Petri net process model. Next, the implementation to add long-term dependency on the process model from last step is shown. At last, another feature is displayed to show the brief evaluation result based on confusion matrix.

2.1 Inputs And Outputs

The plug-in inputs are an event log L with labels to classify each trace and an existing model N possible in multiple forms.

- Acceptable event log L with labels. Labels are one trace attribute and used to identify the positive and negative instances.
- Acceptable Process Models
  - Petri net + Initial Marking.
  - Accepting Petri net.
    The initial marking is already included into the accepting Petri net.
  - Petri net.
    In this situation, the initial marking is guessed automatically in the background program.

The outputs are one process model and its corresponding initial marking. They are exported from the control panel in the result view and can be in various forms.

- Petri net with long-term dependency after Reducing Silent Transition
- Petri net with long-term dependency
- Petri net without long-term dependency
- Process tree

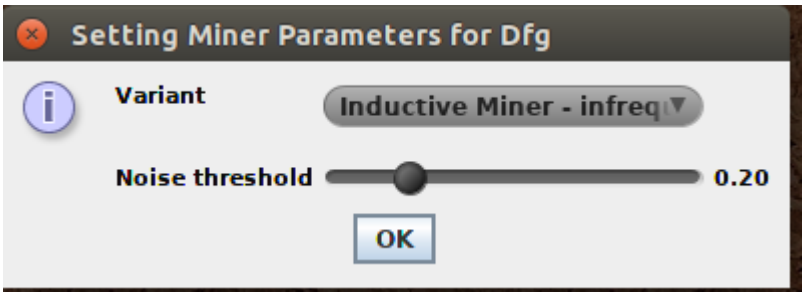


Figure 2.1: Inductive Miner Parameter Setting

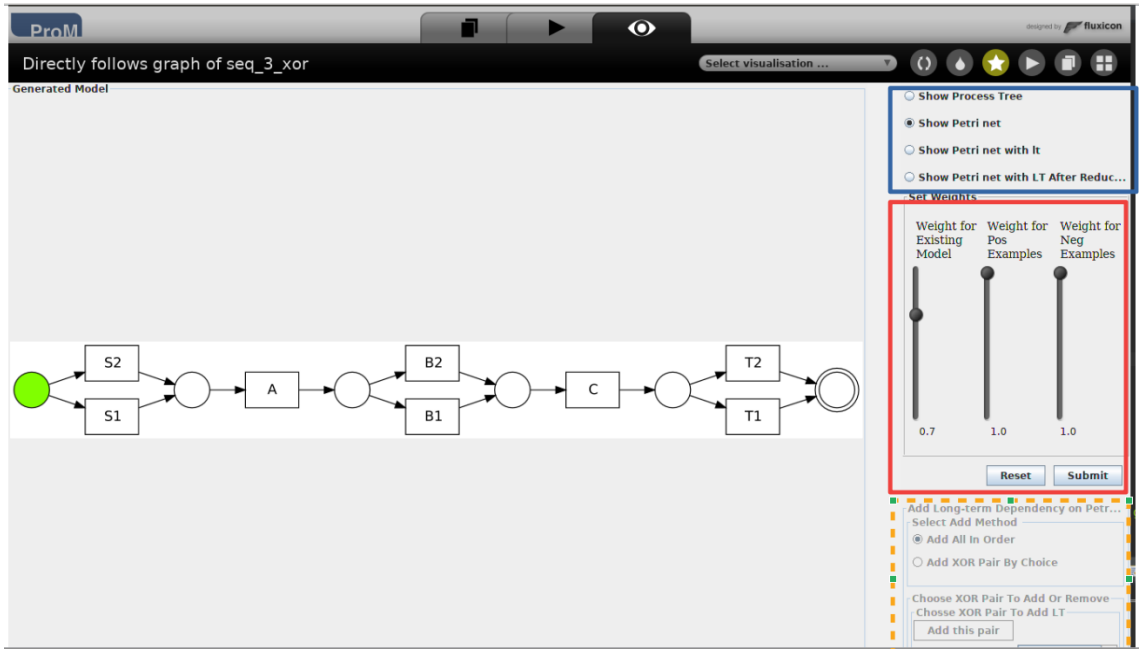


Figure 2.2: Generated Petri net without long-term dependency

2.2 Implementation of dfg-method

Firstly the dialog for options to generate directly-follows graphs from event log pops up. Event classifier are set by those dialogs. Subsequently, a dialog is shown to set the Inductive Miner parameters. The parameters include the Inductive Miner variant and the noise threshold to filter the data. The dialog is displayed in Figure 2.1.
After setting the parameters, process models of process tree and Petri net without long-term dependency can be generated by Inductive Miner and displayed in the result view in Figure 2.2. The left side is the model display area. To allow more flexibility, this plug-in are interactive by the control panel, which is the right side of result view. Originally, only the generated model type and the weight sliders are enabled, while the control panel for adding long-term dependency are invisible.

The model type are in the blue rectangle marked in Figure 2.2. It has 4 options to control the generated model type. Currently, the option "Show Petri net" is chosen, so the constructed model is Petri net without long-term dependency. The weights sliders are in red rectangle. It enables to adjust the weights on the existing model, positive

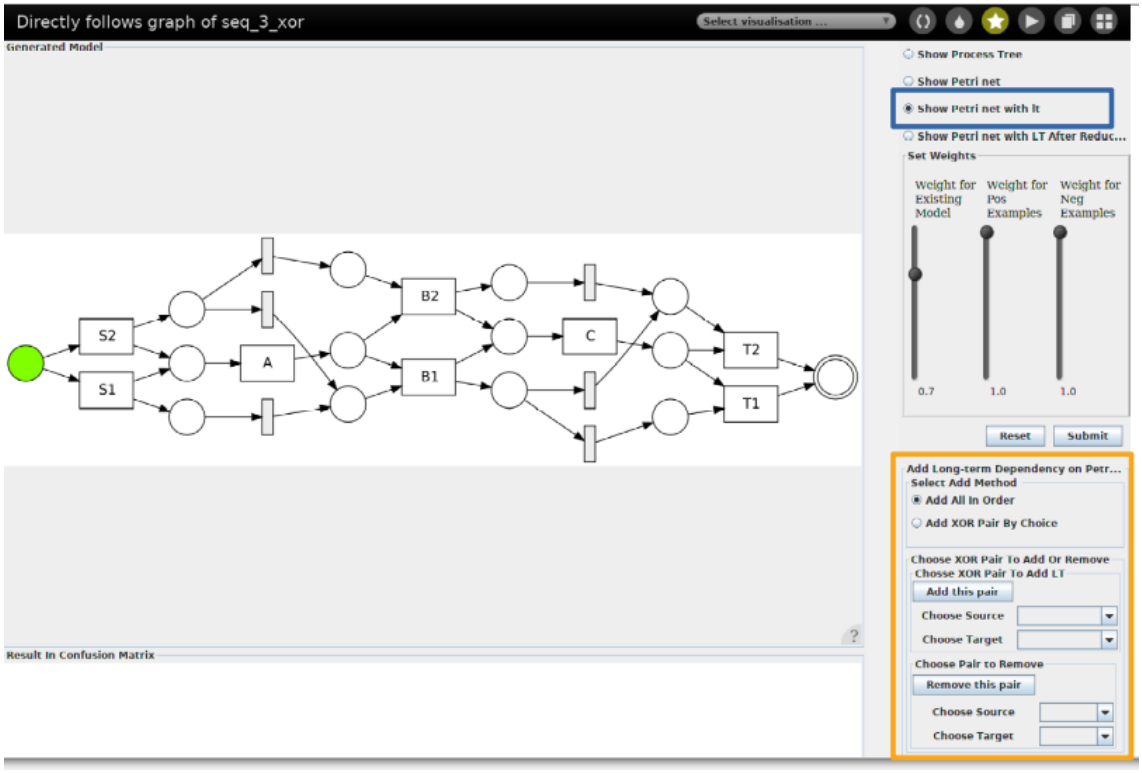


Figure 2.3: Petri Net with long-term dependency

and negative instances. Once submitted those options, different process models are mined under different weights. The rectangle in orange are the invisible part to control long-term dependency options. It is discussed in the next section.

2.3 Implementation of Adding Long-term Dependency

If the model ro generate is Petri net with long-term dependency, the program to add long-term dependency is triggered. This program in the background detects and puts places and silent transitions on Petri net directly mined from Inductive Miner to add long-term dependency. As comparison, the same weight setting is kept like the Figure 2.2, but the option to show a Petri net with long-term dependency is chosen. The resulted model is Figure 2.3.

Meanwhile, the control part of adding long-term dependency turns visible, which is in the orange rectangle in Figure 2.3. It has two main options, one is to consider all long-term dependency existing in the model, the other is to choose the part manually. It allows more flexibility for users. Below those two options, it is the manual selection panels, including control part to add and remove pair. As an example, the blocks Xor(S1,S2) and Xor(T1,T2) are chosen to add long-term dependency. It results in the model in Figure 2.4.

By choosing *Petri net with LT After Reducing* in model type option panel, silent transitions are reduced to simplify the model. Under the same setting in Figure 2.2, the simpler model in Figure 2.5 is constructed, after the post processing of reducing silent transitions.

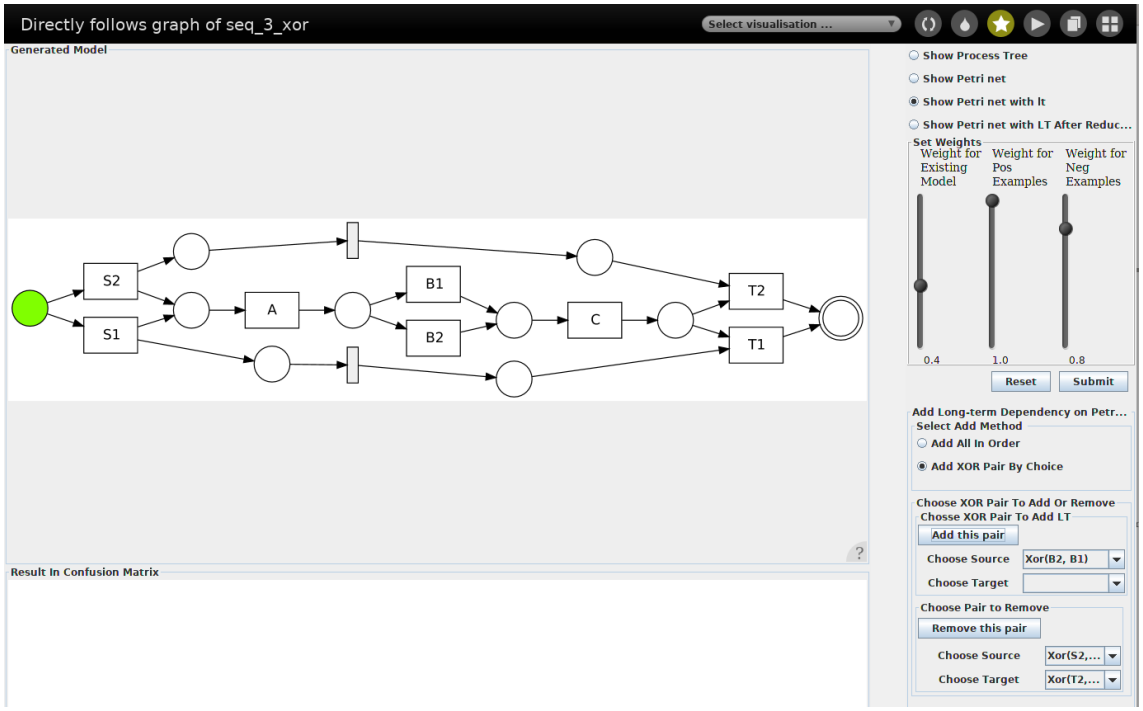


Figure 2.4: Petri net with selected long-term dependency

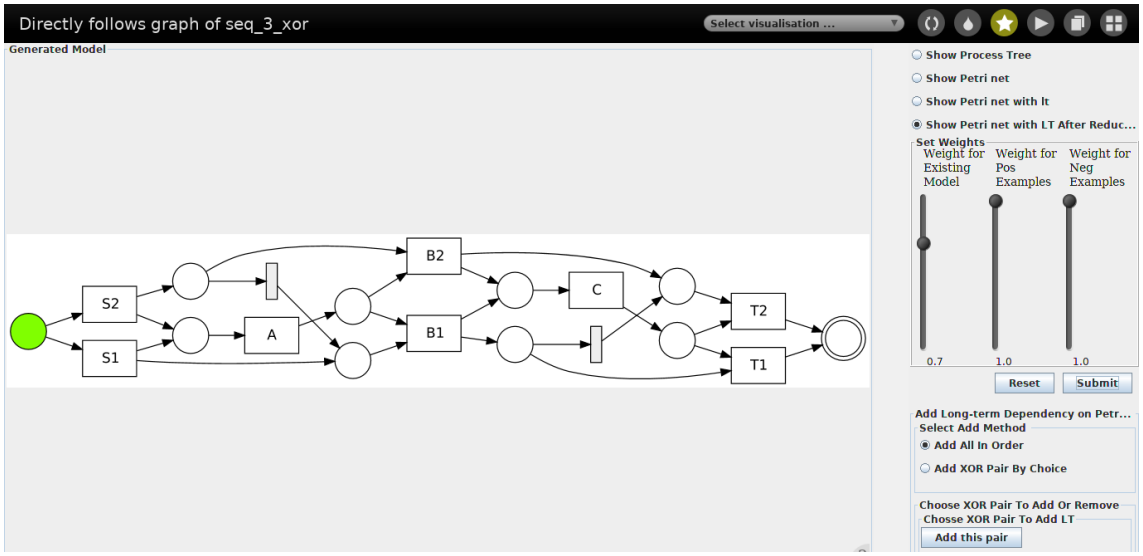


Figure 2.5: Petri net after reducing the silent transitions

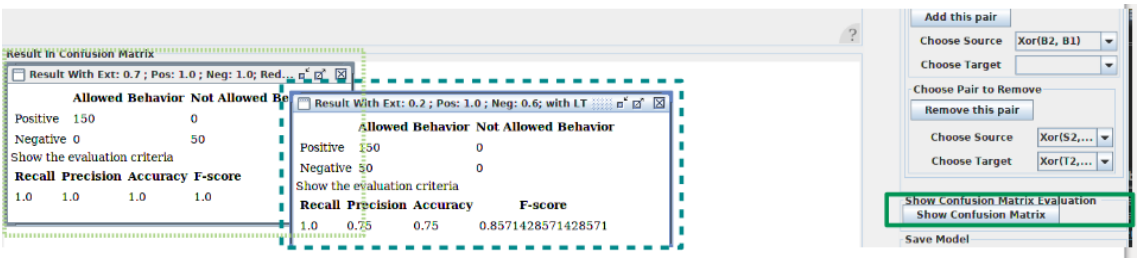


Figure 2.6: Generated Process Tree Model

2.4 Implementation of Showing Evaluation Result

Another feature in this plugin is to show the evaluation result based on confusion matrix. With the brief evaluation result, it helps set the parameter and select the final process model.

It works in this way. After creating the current model in the left view, the evaluation program in background uses the event log and the current Petri net in the view as inputs. It applies a naive fitness checking and generates a confusion matrix with relative measurements like recall, precision. This evaluation result is then shown in the bottom of the left view in Figure 2.6. If the button of green rectangle in the right view *Show Confusion Matrix* is pressed again, the program is triggered again and generates a new confusion matrix result in dark green dashed rectangle which will be listed above the previous result in light green dashes area.

Bibliography

[1] 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.

[2] Kefang Ding. Incorporatenegativeinformation.

[3] 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.

[4] Eindhoven Technical University. © 2010. Process Mining Group. Prom introduction.

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

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

[7] 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.