This work was submitted to:

Chair of Process and Data Science (PADS - Informatik 9), RWTH Aachen University

Translucify: Enhancing Event Logs with Enabled Activities Beyond the Control-Flow Perspective

Bachelor's Thesis

Author: **Geonho Yun**

Student ID: **422305**

Supervisors: Harry H. Beyel
Christopher T. Schwanen

Examiners: Prof. Wil M. P. van der Aalst
Prof. Two

Registration Date: YYYY-MM-DD

Submission Date: YYYY-MM-DD

Abstract

Abstract

This is *not* the final version of the thesis paper, but a draft version needed for the theis registration. The abstract will be updated in the final version of the thesis paper.

Contents

Chapter 1

Introduction

Process mining is a field of computer science aiming to reconstruct the latent process model from a manifest event data. Together with the surge of data availability, process mining has become a hot topic in the fields of data science and business intelligence.

The foundation of all process mining discovery algorithms is the so-called *event log*: A chronologically ordered list of every event recorded in a business system. Event logs have three mandatory feature attributes. The *Case* represents which case the event belongs to. *Activitiy* portrays what kind of event happened. *Timestamp* logs the time of occurrence of that specific event.

(Insert an example event log here)

The three major branches of process mining are *process discovery*, *conformance checking*, and *process enhancement*. *Process discovery* is where a process model is extracted from a submitted event log. In *conformance checking*, the quality of the discovered process model is evaluated in terms of various criteria such as fitness, precision, generalization, and simplicity. Finally, in the *enhancement* phase, the idea is to extend or improve the existing process model using the data present in the event log.

1.1 Motivation

Business processes are seldom linear. Instead, they are usually a messy, chaotic, intertangled mash of activities where its golden path is meticulously hidden. On top of that, event logs have no guarantee of completeness. The quality of the process model produced by process-discovery techniques is therefore entirely dependent on the quality of its event log.

In an ideal world where perfect process discovery is conceivable, an event log would be *transparent*, containing metadata of structural properties of the corresponding process model, e.g., state information in a Petri net setting. Event logs, however, are usually *opaque* - one cannot identify the underlying process model straight away by solely looking at the log. One must instead utilize process-discovery algorithms to generate corresponding models. By its nature, process event logs primarily focus on what *happened*. However, they often do not take into consideration what *could have happened* instead. An event log is *translucent* if the log contains the information which alternative activities could have potentially taken place instead of the actual activity occurred in the real world. Logs of this nature are called *translucent event logs* and are extremely beneficial to enhance the quality of existing process-discovery algorithms.

Let us consider a small example as motivation. Suppose the underlying model of our business process is represented as the Petri net below.

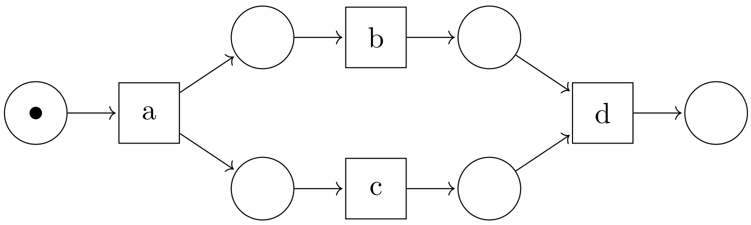

Figure 1.1: Example business model represented as a petri net.

We play-out the model fifty times and retrieve the event log $\mathcal{L}_1 = [\langle a, b, c, d\rangle^{48}, \langle a, c, b, d\rangle^{2}]$ in the process. We can then leverage widely used process-discovery algorithms to rediscover the process model described in Figure 1.1.

This works under the assumption of log completeness, but what would happen if the trace $\langle a, c, b, d\rangle$ occurs so rarely that we weren't able to capture the behavior? Given the log subset $\mathcal{L}_2 = [\langle a, b, c, d\rangle^{50}]$, every process-discovery algorithm will return a linear process model. The translucent variant would look like $\mathcal{L}_3 = [\langle \underline{a}, \underline{b}c, \underline{c}, \underline{d}\rangle^{50}]$, where all enabled activities are listed and the executed activity is underscored. Here, the choice situation between activities b and c is clearly visible, thus preventing the linear modelling of our process.

Despite its benefits, lucent models and translucent event logs are relatively new concepts and are therefore scarcely researched. As a result, translucent event logs are hardly available in real-life process logs. This motivates us to devise novel methods to generate translucent event logs from a non-translucent event log.

1.2 Problem Statement

1.2.1 Limitations of Previous Methods

Insufficient Previous Methods

The problem of embedding translucent information in the event log is relatively new and few methods have been proposed so far. This prompts us to explore new methods to annotate event logs with enabled activities.

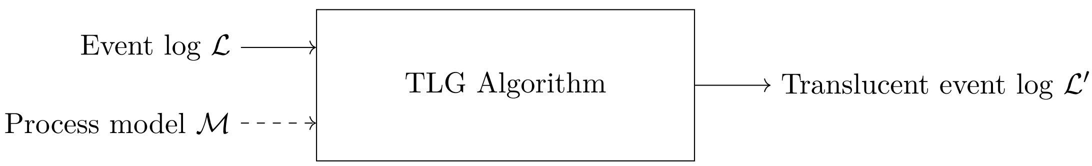

Figure 1.2: Workflow of Translucify. The algorithm takes an event log $\mathcal{L}$ and and optional process model $\mathcal{M}$, then returns an annotated translucent event log $\mathcal{L}'$.

Limitations of Previous Methods

Among the previously suggesting methods, one of them involves annotating the event log with activities which are in turn pattern-matched by labeling the user's system interface. This requires manual labour of labeling individual patterns and are rather unrealistic for real-life systems with a sizable variety of system interfaces.

Another method suggests replaying the event log on the provided process model and annotating the enabled activities in the log. This method is more feasible, but the quality of the annotation, i.e., the accuracy of the enabled activities, is heavily dependent on the quality of the process model. As process models returned by process-discovery algorithms such as the Inductive Miner tend to be underfitting, the method above will likely take a superset of enabled activities for each event, thereby reducing its accuracy.

Neglecting the Data-Centered Aspect

Current methods of translucent log annotation are solely focused on the control-flow aspect, even though the data attributes of event logs also contain valuable information and have an impact on activity enablement.

1.2.2 Problem Declaration

Taking these aforementioned aspects into account, our problem statement can be formulated as the followng.

Translucent Log Extension: *Given an event log and an auxillary process model, annotate the event log with translucent information while incorporating the data attributes into the computation.*

1.3 Research Questions

In the scope of the thesis, we define the five research questions below:

RQ1. Which techniques can detect enabled activities considering solely the log?

RQ2. Which techniques can detect enabled activities considering the log-model pair?

RQ3. How do these methods compare to each other in terms of accuracy and runtime?

RQ4. How can we design and implement an intuitive, user-friendly tool to demonstrate our results?

1.4 Research Goals

The principal resarch goal of this thesis paper is to explore different methods to annotate event logs with translucent information. Furthermore, by building a meaningful, easily operable end-user framework dedicated to translucent log annotation, the **Translucify** framework should be able to evaluate these methods systematically with respect to its accuracy and runtime.

We hereby explicitly state that while the generated translucent log could be applied for further use, especially for process discovery to enhance previously discovered models, the exact application of generated translucent event logs is not the scope of this thesis. Instead, our sole focus lies on the log enhancement phase.

1.5 Contributions

Chapter 2

Preliminaries

In the upcoming section, we will introduce basic mathematical foundations essential for the thesis. These encompass the definitions of sets, multisets, sequences, Petri nets, and event logs. We will also introduce the concept of transition systems and prefix automata. Lastly, we will provide a cursory explanation of machine learning algorithms such as Multivariate Regression as well as deep learning algorithms such as the Transformer architecture.

Definition 2.1 (Set)**.** A set is a collection M of distinct objects. The objects in a set are called *elements* of the set. The set M is denoted as $M = \{a_1, a_2, \dots, a_n\}$, where $a_1, a_2, \dots, a_n$ are the elements of the set. $|M|$ denotes the cardinality of M, i.e. the number of elements of M.

Throughout the thesis, the set of natural numbers is denoted as $\mathbb{N} = \{0, 1, 2, 3, \dots\}$.

Definition 2.2 (Multisets and sequences)**.** The set of all multisets of a set A is denoted with $\mathbb{B}(A)$. $\sigma = \langle a_1, a_2, \dots, a_n \rangle \in A^*$ denotes a sequence over A of length $|\sigma| = n$.

2.1 Process Mining

2.1.1 Petri Nets

Definition 2.3 (Petri net)**.** Let P, T be finite, disjoint sets, where T is a set of *places* and T set of *transitions*. A *Petri net* is a triple $N = (P, T, F)$, where $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs between places and transitions.

Petri nets are the standard process model used in process mining, whose biggest advantage is the ability to model concurrent systems. Places of a Petri net correspond to states of a process, whereas transition of a Petri net correspond to event activities. In order to portray this behavior, we label transitions with activity names.

Definition 2.4 (Marked Labeled Petri net)**.** Let $N = (P, T, F)$ be a Petri net. A *labeled Petri net* is an extended tuple $N = (P, T, F, \mathcal{A}, l)$, where $\mathcal{A}$ is a set of activity labels and $l : T \to \mathcal{A}$ is a labeling function. A *marked Petri net* is an ordered pair (N, M) where $M \in \mathbb{B}(P)$ is a multiset over P.

There are cases where a transition does not correspond to any of the activity names in the event log, as some are only used to model the control flow of a Petri net. These transitions are called *silent transitions* (also τ-transitions) and are denoted with the activity name τ.

Definition 2.5 (Enabled transition)**.** Let (N, M) be a marked Petri net. We further define the $\bullet$ notation, where ${}^\bullet x = \{y \mid (y, x) \in F\}$ and $x^\bullet = \{y \mid (x, y) \in F\}$.

A transition $t \in T$ is *enabled* in M if and only if ${}^\bullet t \leq M$. We also characterize this property using the notation $(N, M)[t\rangle$. The set of enabled transitions in a marked Petri net (N, M) is denoted as $en(N, M) = \{t \in T \mid {}^\bullet t \leq M\}$.

An enabled transition can be fired, which removes a token from each of its input places and adds a token to each of its output places.

Definition 2.6 (Firing rule)**.** Let (N, M) be a marked Petri net. The firing of a transition $t \in T$ is denoted as $(N, M)[t\rangle(N, M')$, where $M' = (M / {}^\bullet t) \cup t^\bullet$.

Let $\sigma = \langle t_1, t_2, \ldots, t_n\rangle \in T^*$ be a sequence of transitions. $(N, M)[\sigma\rangle(N, M')$ denotes that there exists a sequence of markings $\langle M_1 = M, M_2, \ldots, M_{n+1} = M'\rangle$ so that $(N, M_i)[t_i\rangle(N, M_{i+1})$ for all $1 \leq i \leq n$. The set of all reachable markings from M is denoted as $[N, M\rangle = \{M' \mid (N, M)[\sigma\rangle(N, M')\}$ for some $\sigma \in T^*$.

(Show example of a marked Petri net and the firing sequence)

Definition 2.7 (Lucency)**.** Let (N, M) be a marked Petri net. (N, M) is *lucent* if and only if:

$$\forall M_1, M_2 \in [N, M\rangle : en(N, M_1) = en(N, M_2) \Rightarrow M_1 = M_2.$$

Definition 2.8 (Stoachastic Data Petri net)**.** Let $N = (P, T, F, \mathcal{A}, l)$ be a labeled Petri net. A tuple $(P, T, F, \mathcal{A}, l, \omega)$ is *stochastic* if and only if $\omega : T \times \Delta \to \mathbb{R}^+$ is a weight function.

Given a state $p \in P$ and a data state $\delta \in \Delta$, the probability that the Petri net N fires a transition $t \in p^\bullet$ is given as the following equation:

$$p(t \mid p, \delta) = \frac{\omega(t, \delta)}{\sum_{t' \in p^\bullet} \omega(t', \delta)}.$$

2.1.2 Event Logs

Definition 2.9 (Event)**.** Let $\mathcal{E}$ be the event universe. An *event* $e \in \mathcal{E}$ is a logical abstraction of a real-life process event. An event possesses multiple named attributes. We define the universe of all attribute names as $\mathcal{AN}$ and the universe of all attribute values as $\mathcal{AV}$.

Based on this, we define the attribute projection function $\pi : \mathcal{E} \times \mathcal{AN} \to \mathcal{AV} \cup \{\perp\}$, where π is a partial function mapping the attribute name of every event to an attribute value (otherwise a none value $\perp$). Following the convention in [1], we denote the signature $\pi(e, n)$ as $\pi_n(e)$ for all $e \in \mathcal{E}, n \in \mathcal{AN}$.

Subsequently, a collection of events form an event log of a system process.

Definition 2.10 (Event Log)**.** Let $\mathcal{C}, \mathcal{A}, \mathcal{T} \subseteq \mathcal{AV}$, where $\mathcal{C}$ is the universe of case identifiers, $\mathcal{A}$ the universe of activity names, and $\mathcal{T}$ the universe of timestamps. An event log $\mathcal{L} \subseteq \mathcal{E}$ is a subset of the event universe such that for all events $e \in \mathcal{L}$:

- $\pi_{case}(e) \in \mathcal{C}$ is the case identifier,
- $\pi_{act}(e) \in \mathcal{A}$ is the activity name,
- $\pi_{time}(e) \in \mathcal{T}$ is the timestamp.

We further assume that in a conventional event log, there exists a total order $<_{time}$ on $\mathcal{L}$ such that for all $e, e' \in \mathcal{L} : e <_{time} e' \Leftrightarrow \pi_{time}(e) < \pi_{time}(e')$, i.e. the events are sorted by their timestamps in chronological order.

We can further group events by their case identifiers. A sequence of events with the same case identifier ordered by $<_{time}$ is called a *trace*. Note that the set of all traces of an event log is pairwise disjoint, i.e., there is no event $e \in \mathcal{E}$ which is an element of two different traces. Hence, an event log can also be represented as a set of traces.

Definition 2.11 (Trace)**.** Let $\mathcal{E}$ be the event universe. A *trace* is a sequence of events $\sigma = \langle e_1, e_2, \ldots, e_n \rangle \in \mathcal{E}^*$ such that for all $i \in \{1, \ldots, n-1\} : \pi_{case}(e_i) = \pi_{case}(e_{i+1})$.

However, when discussing about traces, we oftentimes refer to them as a sequence of activities. This is firstly for the sake of simplicity, but also due to the fact that the control flow is considered to be the most crucial aspect in process mining, in particular when constructing process models. Event logs where each trace is solely represented as a sequence of activities is called a *simple event log.*

Definition 2.12 (Simple Event Log)**.** Let $\mathcal{L}$ be an event log and $\sigma \in \mathcal{L}$ a trace. We expand the attribute projection function analogously for traces as the following: $\pi_n(\sigma) = \pi_n(\langle e_1, e_2, \ldots, e_n \rangle) = \langle \pi_n(e_1), \pi_n(e_2), \ldots, \pi_n(e_n) \rangle$. $\mathcal{L}'$ is a simple event log of $\mathcal{L}$, if:

$$\mathcal{L}' = \bigcup_{\sigma \in \mathcal{L}} \pi_{act}(\sigma).$$

Since we are projecting only the activity names of each event, we lose the uniqueness of each event, resulting in losing the uniqueness of each trace as well. In the simple event log setting, we therefore need to represent the event log as a multiset of traces.

The objective of our thesis is to transform conventional event logs into translucent event logs by annotating each event $e \in \mathcal{E}$ with an additional attribute en. en specifies all activities which could have happened the moment $\pi_{act}(e)$ occurred. We formally define the notion of translucent event logs below.

Definition 2.13 (Translucent Event Log)**.** Let $\mathcal{L}$ be an event log. $\mathcal{L}$ is *translucent* if and only if for all $e \in \mathcal{L} : \pi_{en}(e) \subseteq \mathcal{A}$ and $\pi_{act}(e) \in \pi_{en}(e)$.

(Example of a translucent event log)

2.1.3 Transition Systems

Definition 2.14 (Transition System)**.** Let S be the set of states, A the set of activities, and $T \subseteq S \times A \times S$ the set of transitions. A *transition system* is a triple $TS = (S, A, T)$. We denote the set of initial states as $S^{start} \subseteq S$ and the set of final states as $S^{end} \subseteq S$, where $S^{start}, S^{end} \subseteq S$.

Transition systems are an alternative method next to Petri nets to represent processes of a system. Due to its structural property, silent transitions are absent in transition systems.

This relieves us from the need to compute alignments when replaying the event log on the model.

Definition 2.15 (Prefix Automaton)**.** Let $\mathcal{L}$ be an event log and $l^{state} : \mathcal{L} \times \mathbb{N} \rightarrow S$ a state representation function. $TS_{L,l^{state}} = (S, A, T)$ is a transition system based on $\mathcal{L}$ and l^{state} with the following properties:

- $S = \{l^{state}(\sigma, k) \mid \sigma \in \mathcal{L} \wedge 0 \leq k \leq |\sigma|\}$ the state space,
- $A = \{\sigma(k) \mid \sigma \in \mathcal{L} \wedge 1 \leq k \leq |\sigma|\}$ the set of activities of the event log,
- $T = \{(l^{state}(\sigma, k), \sigma(k+1), l^{state}(\sigma, k+1)) \mid \sigma \in \mathcal{L} \wedge 0 \leq k < |\sigma|\}$ the set of transitions,
- $S^{start} = \{l^{state}(\sigma, 0) \mid \sigma \in \mathcal{L}\}$ the set of initial states, and
- $S^{end} = \{l^{state}(\sigma, |\sigma|) \mid \sigma \in \mathcal{L}\}$ the set of final states.

(Show example of a prefix automaton)

Frequently used examples of a state representation function take the prefix function hd^k as its baseline mechanism. Among diverse methods of process representation using transition systems, we focus on the list, set, and multiset representations.

(Show example of the set and multiset variant of previous example)

All these three variants have their own advantages and disadvantages. The list representation is the most precise, as it preserves the activity order of each trace. However, analyzing parallel situations is difficult due to the very same characteristics of order preservation. Moreover, the representation fails at recognizing loops in the inherent process model and is therefore susceptible to overfitting.

The set representation, on the other hand, is more general and does not suffer from order preservation issues as the list representation does. Although the property of order negligance is practical for local loops and parallel situations, it is critical for larger loops. Consider e.g. nested loops or a loop situation where all activities have been already visited in the first iteration. This would lead to drastic state simplification and the resulting information loss. here, it is crucial to select an optimal prefix length k.

The multiset representation is a compromise between the list and set representations. However, it is still not able to correctly capture the loop behavior, as each iteration would create a new set of states.

2.2 Machine Learning Algorithms

2.2.1 Conventional Machine Learning Algorithms

Random Forest

Logistic Regression

2.2.2 Deep Learning Algorithms

Transformer Architecture

Chapter 3

Related Work

3.1 Data-Aware Process Mining

Conventional process discovery algorithms only consider the control-flow aspect of the process, i.e., the activity attribute of the event log, thereby ignoring the data attributes the event log provides. Data-aware process mining, on the other hand, attempts to incorporate both the control-flow and data attributes of the event log. The prevalent repretoire is to discover decision points in the model and to annotate them with guard functions, which in turn are discovered with decision trees. This method of process enhancement is called *decision mining* [2–4], and is a well-estalished field of research within process mining. [4] proposes a Petri net with data *(DPN-net)* setting to expand the notation of Petri nets. While previous papers worked with the assumption of deterministic, mutual exclusive transition behavior in decision points, they do not take into account how certain decisions cannot be modeled dichotomously. Often, one needs a softer classification assumption, stating that data attributes affect the decision probabilistically. [5] further extends the concept of data-annotated Petri nets by integrating stochastic information into the model. In the paper, the authors introduce the concept of stochastic labeled data Petri nets *SLDPNs* and propose a method to generate an SLDPN from a Petri net and an event log. Each transition will be mapped with its own weight function learned with the activiation instances of individual transitions using logistic regression.

3.2 Translucent Event Logs

Little research has been performed on the topic of translucent event logs. Being a relatively young concept in the field of process mining, they were first hinted in 2018 by [6], where a possible event log revealing the set of enabled activities is mentioned. [7] formally introduces and defines translucent event logs and relates concepts of lucency and translucency by showing that a lucent process model can be rediscovered by using a translucent event log retrieved from the model. Methods of creating translucent event logs are first discussed in [8]. Here, a system's screenshot is matched with the labelled activity pattern and annotated with the corresponding activities. Furthermore, a model-based approach is introduced by replaying the event log on the model and annotating enabled activities. [9] formally introduces a precision measure between a Petri net and a translucent event log by comparing log-enabled activities and model-enabled activities.

3.3 Predictive Process Monitoring

- Explain the term and list previous differnet approaches
- Lin et al. [10]: Next event and next data attribute prediction using RNNs
- Gunnarsson et al. [11]: Remaining trace and runtime prediction using LSTMs
- Bukhsh et al. [12]: Next activity, next event time, and remaining time prediction using transformer architecture
- General overview of the field is given in [13].

Chapter 4

Translucifying the Event Log

4.1 Framework Overview

In order to look deeper into the main problem of this thesis, we first formally define our problem as described below.

Definition 4.1 (Translucent Log Extension Problem)**.** Given an event log $\mathcal{L} \subseteq \mathcal{E}$ as input, produce a translucent event log $\mathcal{L}'$ where the set of enabled activities are added as attributes.

There is a variant of the *Translucent Log Extension Problem*, where a process model is provided along with the event log.

Definition 4.2 (Translucent Log Extension Problem - Process Model Variant)**.** Given an event log $\mathcal{L}$ and a process model $\mathcal{M}$ as inputs, produce a translucent event log $\mathcal{L}'$ where the set of enabled activities are added as attributes.

The second variant differs from the first, as the model is meant to act as a reference model; the set of enabled activities is intended to be constrained by the process model. Note that the function of $\mathcal{M}$ is to provide an upper bound on the set of enabled activities, and the log is there to provide further constraints to enrich the model. Of particular interest are parallel and choice situations, since we are able to deduce supplementary patterns not demonstrated in the process model using log data. We name the first variant defined in Definition 4.1 as *Bottom-up Translucent Log Extension Problem*, whereas the second variant in Definition 4.2 is named as *Top-down Translucent Log Extension Problem.*

4.2 Top-down Approaches

4.2.1 Petri Net-based Approaches

The basic notion of Petri nets was introduced in the previous chapter. . . .

Given a Petri net and an event log, we utiltize the alignment-based appraoch presented in [8] as its baseline algorithm. After computing the alignment for each trace, the program replays the alignment on the reachabiliy graph trace by trace in order to circumvent the silent transitions. For each activity, the algorithm then augments the event log with the

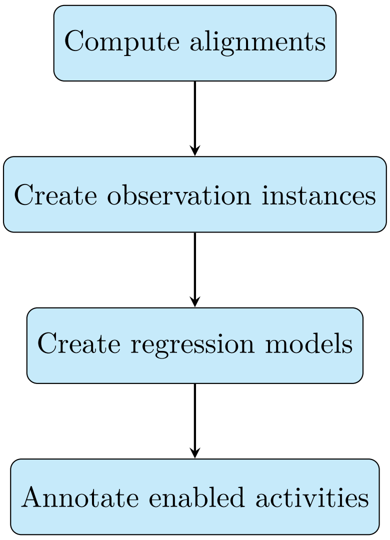

Figure 4.1: Stages of multivariate regression on an SLDPN.

corresponding transitions situated in outgoing arcs of the current state. After creating a basic translucent event log, the log can be refined by further algorithms.

...

When annotating the event log with enabled activities, it might make sense to assume that not all activities enabled in the model are enabled in reality. The model might be too permissive to guarantee its fitness to the event log, and the set of enabled activities depicted in the model should therefore be considered as an upper bound of the actual set. Here, instead of applying a hard-line policy by adding guards to the transitions as done in the field of decision mining, we can utilize a probabilistic approach to filter out the activities. An important question would be how we should compute the transition probability of the model. This is where the SLDPN model comes into play.

A method to incorporate the data attributes is to implement multivariate regression. Similar to the setting in [5], we construct a training data set for each transition of a Petri net consisting of data attributes and a boolean label indicating whether the transition was executed given the data attributes as input. We then perform a regression analysis on the training data set for each transition. The resulting dictionary of transitions and regression functions can be employed to filter out transitions lying below a certain probability threshold p in each decision point during replay.

Table 4.1: Example event log for multivariate regression.

Case ID	Activity	Timestamp	Family History	Amount
101	a	2024-08-01 08:00:00	yes	30
101	d	2024-08-01 08:30:00	yes	50
102	a	2024-08-02 09:00:00	no	0
102	b	2024-08-02 09:45:00	no	40
102	d	2024-08-02 10:30:00	no	56

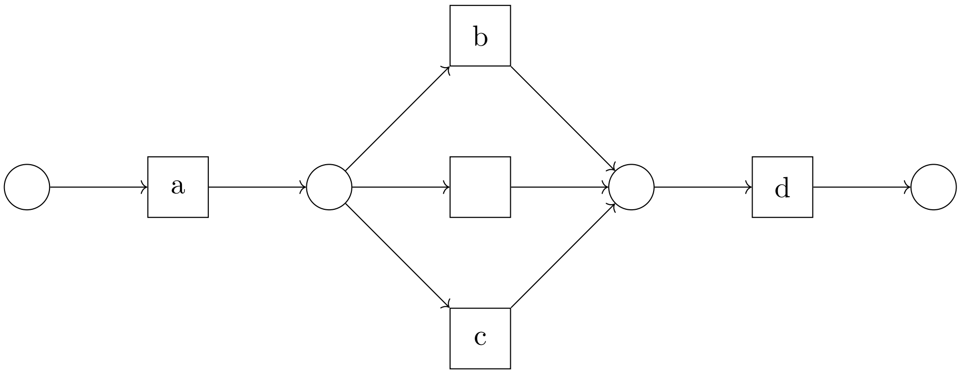

Figure 4.2: Example Petri net for multivariate regression.

Computing Alignments

A process model does not always guarantee a perfect fitness of the event log. Sometimes, infrequnet traces are considered as noise and are subsequently ignored. Furthermore, process models contain more than one unique fitting path for each trace in various occasions. It is therefore crucial in Petri nets to first compute the best-fitting model trace for each log trace using *alignments*.

(theoretical background & explanation on how alignments usually work)

Computing alignments can be thought as choosing an alignment function $f_{align} : \mathcal{L} \rightarrow T^*$, where given a log trace $\sigma \in \mathcal{L}$ as input, f_{align} returns a sequence of transitions $(t_1, \ldots, t_n)$ as output, corresponding to the optimal model trace computed by a chosen alignment algorithm.

In our scenario, however, we need to take data attributes into account as well, which are necessary for the regression analysis. For this, the appropriate data attributes of the log trace must be attached to the model trace. Intuitively, since event logs are records of a system, we assume that the data state is the data snapshot immediately after the event occurrence. Moreover, we assume that silent transitions do not modify the data state. Data states of silent transitions in the model trace are therefore identical to the most recently occurred data state of a named transition recorded in the event log.

We therefore extend the alignment function to $f_{align} : \mathcal{L} \rightarrow (T \times \Delta)^*$, where Δ is the set of data attributes found in the corresponding event $e \in \sigma$. Given a log trace $\sigma \in \mathcal{L}$ and the corresponding data attributes $d \in \Delta$, $f_{align}(\sigma) \mapsto ((t_1, d_1), \ldots, (t_n, d_n))$ returns a sequence of transitions as output.

Using the example event log in Table 4.1 and the model in Figure 4.2, the alignment for the trace $\langle a, d \rangle$ would be $((a, (yes, 30)), (t1, (yes, 30)), (d, (yes, 50)))$, and the alignment for trace $\langle a, b, d \rangle$ would be $((a, (no, 0)), (b, (no, 40)), (d, (no, 56)))$.

Creating Observation Instances

Given an aligned model trace, our goal is to generate a labeled training data set for each transition in the Petri net model. In other words, the goal of this stage is to find the function $f_{observe} : T \rightarrow (\Delta \times \{0, 1\})^*$.

We iterate over the event log and replay the Petri net using the model trace $f_{align}(\sigma)$ $\forall \sigma \in \mathcal{L}$. For each Petri net marking M_i where $(N, M_0)[(t_1, \ldots, t_{i-1})\rangle(N, M_i)$ and the corresponding current data state d_i, we first compute the enabled transitions $en(M_i)$ and update the values of $f_{observe}$ as the following: $f_{observe}(t_j) := f_{observe}(t_j) \cup \{(d_i, c)\}$ for all $t_j \in en(M_i)$, where:

$$c = \begin{cases} 1, & \text{if } t_i = t_j \\ 0, & \text{otherwise} \end{cases}$$

Looking at our running example, the observation instance function $f_{observe}$ would be as follows:

$$f : t \mapsto \begin{cases} \Big[((yes, 30), 1), ((no, 0), 1)\Big], & t = a \\ \Big[((yes, 30), 0), ((no, 0), 1)\Big], & t = b \\ \Big[((yes, 30), 0), ((no, 0), 0)\Big], & t = c \\ \Big[((yes, 30), 1), ((no, 40), 1)\Big], & t = d \end{cases}$$

Create regression models

After creating the observation instances, we train a logistic regression model for each transition in the Petri net model. The goal of this stage is to find the function $f_{regress} : T \times \Delta \to [0, 1]$, which returns the probability of a transition being enabled given the data attributes.

Annotating the log with enabled activities

After training a logistic regression model for each transition, the goal of this stage is to utilize the regression functions and the alignments to add enabled activities to events listed in each trace. In other words, we need to compute a function $f_{annotate} : \mathcal{L} \times [0, 1] \to \mathcal{L}'$, where given a log trace $\sigma \in \mathcal{L}$ and a threshold $t \in [0, 1]$, $f_{annotate}$ returns a translucent trace σ' where $\pi_{en}(e) \in \mathcal{P}(\mathcal{A})$ $\forall e \in \sigma$.

4.2.2 Transition System-based Approaches

... One of the advantages of utilizing transition systems over Petri nets is the lack of silent transitions. When trying to replay a trace on a Petri net in the presence of silent transition, the algorithm must decide which path of the firing sequence it should take by computing alignments with a certain cost function, then taking the path with the minimal cost. This can be computationally expensive and time-consuming, especially for larger models where the number of silent transitions is high. Transition systems can benefit us in this regard.

Definition 4.3 (Frequency-Annotated Prefix Automaton)**.** Let $\mathcal{L}$ be a simple event log. A *Frequency-Annotated Prefix Automaton (FAPA)* is a tuple $PA_{freq} = (S, A, T, f)$, where (S, A, T) is a prefix automaton $TS_{L,hd}$ following the definition 2.15 and $f : S \to \mathbb{N}, \sigma_{pref} \mapsto |\{\sigma \in \mathcal{L} \mid \sigma_{pref} \sqsubseteq \sigma\}|$ is a frequency labeling function.

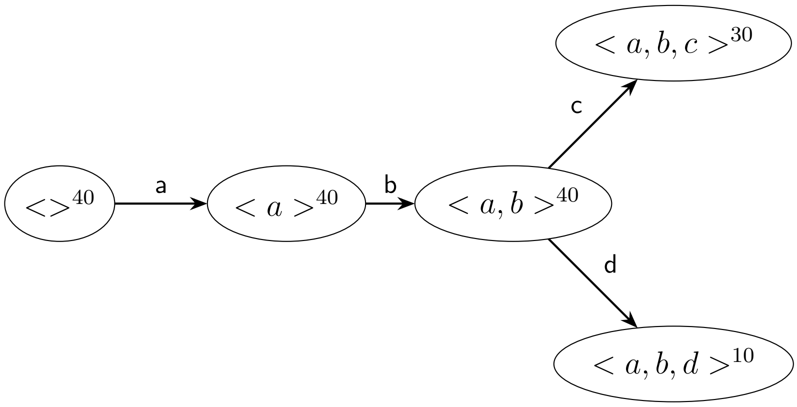

Figure 4.3: Resulting FAPA from $\mathcal{L}$.

Let us look at a small example. Consider the simple event log $\mathcal{L} = [\langle a, b, c\rangle^{30}, \langle a, b, d\rangle^{10}]$. In order to generate an FAPA from $\mathcal{L}$, we can use the following algorithm:

The resulting FAPA is depicted in 4.3. The frequency labeling function f is represented as in the usual superscript notation used in trace multiset of simple event logs.

Given a certain threshold, a simple algorithm would be to iterate over each trace in the event log and annotating the enabled activities with transitions lying above the threshold. For a threshold value of e.g. $t = 0.5$, the resulting simple translucent event log would be: $[\langle \underline{a}, \underline{b}, \underline{c}\rangle^{30}, \langle \underline{a}, \underline{b}, c\underline{d}\rangle^{10}]$. Note that annotating enabled activites using conventional prefix automata can be considered as a special case of this problem with threshold $t = 0$.

4.3 Bottom-up Approaches

Buttom-up approaches aims to discover enabled activities without the help of a process model. ...

A simple approach is to utilize the *directly-follows matrix* of an event log. Given an event log $\mathcal{L}$, we compute the set of unique activities and generate a next-activity matrix $\mathbf{A}$, where the entry $\mathbf{A}_{ij}$ represents the number of times activity j follows activity i. We can then easily transform $\mathbf{A}$ into a probability matrix $\mathbf{A}'$ by dividing each row by the sum of the row. We then filter out the results by a certain threshold p and add the entries surviving the threshold to each event trace. This algorithm can serve as baseline for further extensions. We formally define the directly-follows matrix as follows:

Definition 4.4 (Directly-follows Matrix)**.** Let $\mathcal{L}$ be an event log. Let further $U = \pi_{act}(L) = \bigcup\limits_{\sigma \in \mathcal{L}} \{\pi_{act}(e) \mid \pi_{act}(e) \in \sigma\}$ be the set of unique activities in $\mathcal{L}$. $\mathbf{A}$ is a *directly-follows matrix* of $\mathcal{L}$, if:

- $\mathbf{A}$ is a matrix over $\mathbb{N}_0$ with dimenstion $n \times n$, where $n = |U|$, i.e., the number of unique activities in $\mathcal{L}$.
- Let $\{U_j\}_{j \in J}$ be an indexed family of U with $J = \{1, 2, \ldots, n\}, n = |U|$. $\mathbf{A}_{ij} = \sum\limits_{\sigma \in \mathcal{L}} \Big|\{i \mid 1 \leq i \leq |\sigma|, \sigma(i) = U_i \wedge \sigma(i+1) = U_j\}\Big|$, i.e., the absolute frequency activity U_i is directly followed by activity U_j.

Given the directly-follows matrix as a baseline, there are several applications to extend the matrix. One of the most straightforward approaches is transforming the matrix into a Markov matrix.

Definition 4.5 (Probabilistic Directly-follows Matrix)**.** Let $\mathbf{A} = (\mathbf{a_{ij}})$ be a directly-follows matrix of an event log $\mathcal{L}$. $\mathbf{A}' = (a'_{ij})$ is a *probabilistic directly-follows matrix* of $\mathcal{L}$, if:

$$a'_{ij} = \frac{a_{ij}}{\sum\limits_{k=1}^{n} a_{ik}}$$

- Furthermore, we can utilitze a deep-learning based black-box approach. The issue with implementing supervised learning algorithms is that we need a labeled training dataset. In our case, this would be a preexisting translucent event log, which is unavailable in our setting due to missing enabled activities data. We can cirucmvent the problem by training the model using the next activity information as label, as this information is available in every event log. Since most learning algorithms would not return a single value but an underlying probability distribution of possible outcomes, we can substitute the final *argmax* operation with selecting a threshold p and returning all labels lying above it.

Note that this is not the final list as we are still in the process of selecting new methods. The final list of methods will be updated in the final version of the thesis paper.

Chapter 5

Implementation

The program accepts an event log and an optional process model, e.g. a Petri net, as inputs and returns a corresponding translucent event log as output. Users have the option to select from various methods of log generation, will will be specified below. Mainly, these methods can be classified in two categories: top-down approaches which require a Petri net, and bottom-up approaches which solely need the event log.

In order to provide a user friendly interface, we implement a web application. The following sections describe the software specification, architecture, and its features.

5.1 Software Architecture

5.1.1 Backend

The backend was implemented using the Flask[1] framework. Flask is fitting for our use case as it is a lightweight framework providing a minimalisitic set of features needed to build API calls.

For the database, SQLAlechemy[2] was used as an Object Relational Mapping tool to model the database. It was chosen due to its seamless compatibility with Flask using the Flask-SQLAlchemy extension, and its ease of usage.

The database stores event log entities and their corresponding translucent event log entities. Since the event logs tend to be large, the actual event logs are stored in the file system, while the database only keeps track of the metadata, sucsh as the file path, name, and type of the event logs. Translucnet event log entities have an extra attribute `is_ready` in order to indicate whether the computation is complete. The database schema is depicted in Figure 5.1.

[1] www.flask.palletsprojects.com/en/3.0.x/
[2] www.sqlalchemy.org

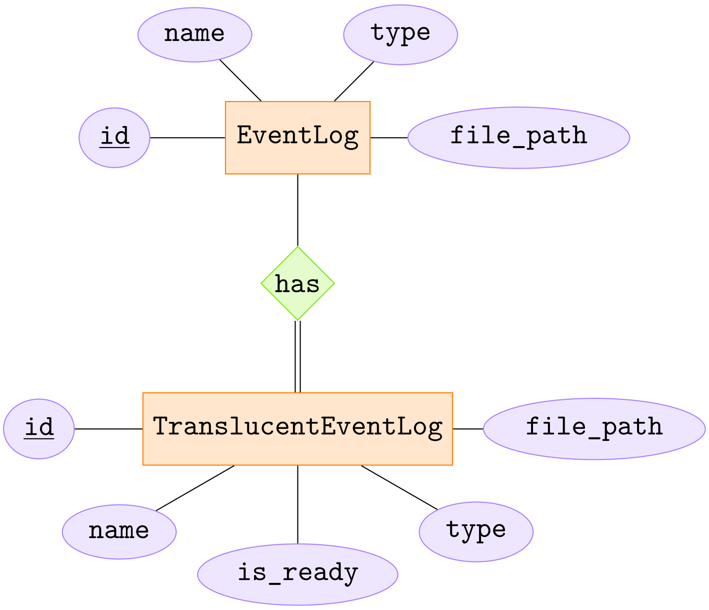

Figure 5.1: Entity-Relationship Diagram of the database schema.

5.1.2 Frontend

React was chosen due to the author's proficiency. The web application will start with an interface where the user can upload an event log, with a choice between CSV and XES. After uploading the event log, different methods of translucent log annotation will be available. The user can select the method and the corresponding necessary parameters. After the log generation is complete, the user can download the log either in a CSV or an XES format.

5.2 Software Features

5.2.1 GPU Computing

For the transformer variant, in order to train the transformer model, a GPU access was necessary due to the high computational requirements. In a local setting, the program would take an extraordinary amount of computing time, or in other cases simply terminate due to insufficient memory.

In order to solve this issue, a connection to the remote PADS HPC Cluster needed to be established. Inside the cluster, a Flask microservice was set up analagously to the local setup. Using SSH tunneling, all requests regarding the transformer model were then forwarded to the remote server. After the computations were finished, the results from the remote server were subsequently delivered back to the main application running in the local server. Figure 5.2 illustrates the SSH port forwarding setup.

5.2.2 Asynchronous Task Distribution

All requests from the frontend are processed by request handlers of Flask. A particular issue we encountered is this case was the excessive computation time for each algorithm run

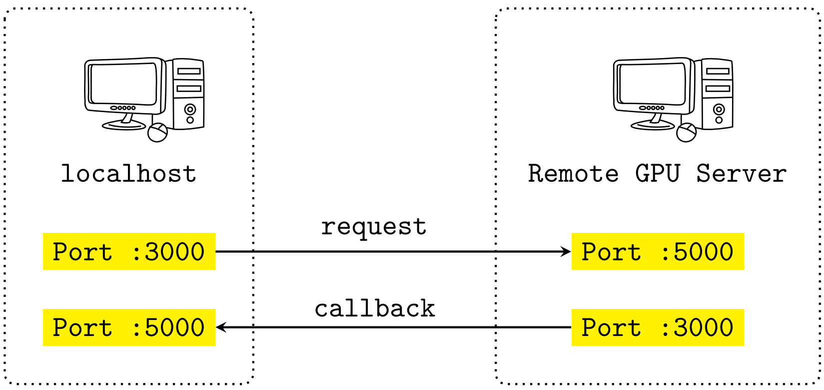

Figure 5.2: SSH Port Forwarding.

in the backend. As a result, when attempting to run the algorithms directly on the request handlers in Flask, the server would block all other requests until the current computation was finished, making the application unresponsive and bluntly unusable. To mitigate this issue, we incorporated a distributed task queue using Redis[3] and Celery[4].

Celery is a distributed system to split up computation in tasks and to assign these to subprocesses, known as workers. Redis is used as a *message broker* so that the Celery client can delegate the tasks to the workers. Upon receiving a request, the Flask server will pass on the task to the Celery client, which will then push the task to a Redis Message Queue. The Celery workeres will then pick up the task from the queue and execute it. Compared to the setup where all tasks are handled directly in the request handlers, this setup allows for a non-blocking behavior of the server, as the tasks are executed in the background by other processes, thereby preventing the application from freezing.

The overall application architecture is depicted in Figure 5.3.

5.2.3 Multivariate Regression With Petri Nets

The sequence is depicted in Figure 5.4.

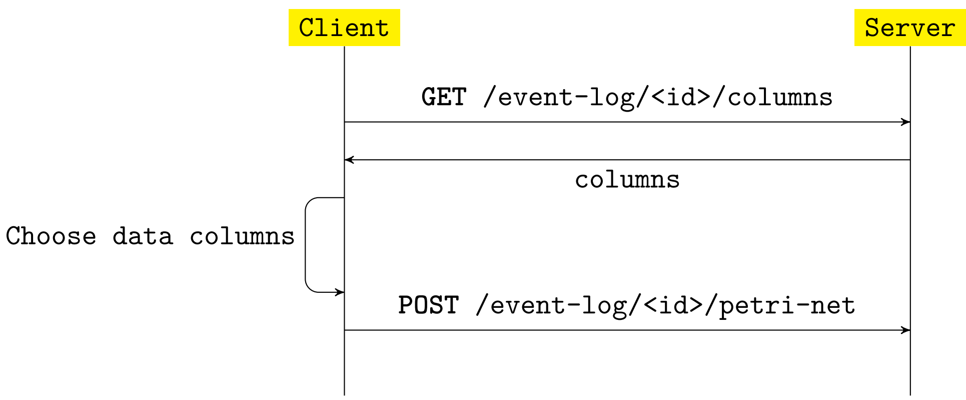

Figure 5.4: Client-server sequence diagram for the Petri net variant.

[3]https://redis.io/
[4]https://pypi.org/project/celery/

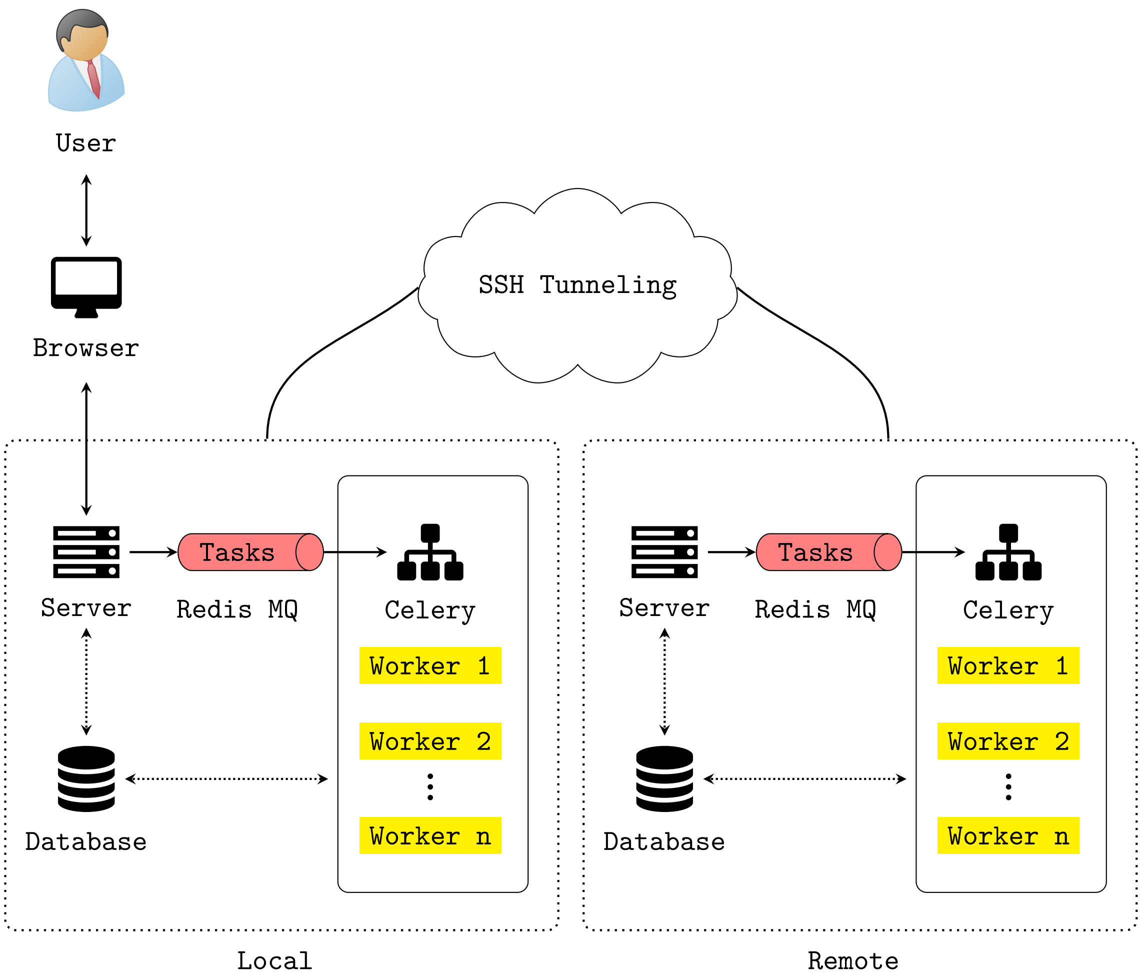

Figure 5.3: Application Architecture

5.2.4 Multivarate Regression with Prefix Automata

The sequence is depicted in Figure 5.5.

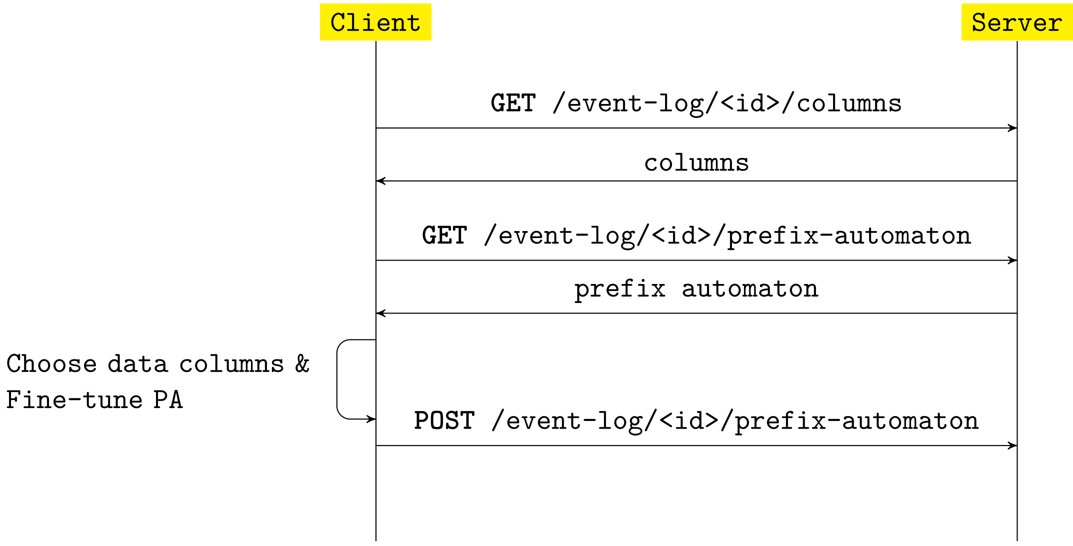

Figure 5.5: Client-server sequence diagram for the Prefix automaton variant.

The implementation of the multivariate regression with prefix automata is done analagously to the Petri net variant.

5.3 Some another section

5.3.1 Transformer Models

The transformer model is implemented using the PyTorch library.

The model is trained on the PADS HPC Cluster using an NVIDIA GeForce RTX 2080 Ti GPU with a total memory of 11264 MiB equipped with CUDA 12.2. Using a learning rate of 0.00001 and a batch size of 16, the model is trained for 50 epochs.

Chapter 6

Evaluation

- Limitations: Using real-life translucent event logs is often implausible for our scenario due to the fact that most real-life logs do not contain the set of enabled activities. Therefore, direct evaluation by receiving a real-life translucent event log as input, stripping away the enabled activities column, inserting the log in our algorithm as input then comparing the result with the original translucent event log is not possible.
- Instead, we can use artificial process models. The evaluation process works like the following:
    1. We generate random process models, e.g. data Petri nets.
    2. We then play-out the model randomly and extract 1. a normal log and 2. a translucent event log.
    3. We then use the normal log as input to our TLG program and compare the result with the original translucent log.
- On top of that, we can also evaluate its versatility by directly comparing models generated using translucent event logs and the usual state-of-the-art process discovery algorithms. The evaluation process works like the following:
    1. Given a normal process log, we use state-of-the-art process discovery algorithms to generate process models.
    2. We use the log as input to our program and generate a translucent event log.
    3. We then generate a process model based on translucent-log based process discovery algorithms.
    4. We then compare the models based on their performance measures.
- In the model annotation setting, we can evaluate the performance of the model extension algorithm by comparing the stochastic precision of the annotated model with the original model. The evaluation process works like the following:
    1. We iterate over each trace and replay it on two models: The original model received as input and the annotated model.

2. For each transition, we calculate the stochastic precision by computing the product of the transition probabilities in each transition step.
3. We then add up the stochastic precision score for each trace and divide it by the total number of traces to get the average stochastic precision score.

Note that this is different from the translucent precision score defined in [9], since we need a precision measure comparable and applicable to both of the original log-model-pair and the translucent-log-annotated-model pair.

- As we are not presenting a single generation method, it will be necessary to compare and evaluate each method separately using the process described above.

Chapter 7

Discussion

Chapter 8

Conclusion

Bibliography

[1] Wil van der Aalst. *Data Science in Action*, pages 3–23. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016. ISBN 978-3-662-49851-4. doi: 10.1007/978-3-662-49851-4_1. URL `https://doi.org/10.1007/978-3-662-49851-4_1`.

[2] A. Rozinat and Wil M. P. van der Aalst. Decision mining in prom. In *Business Process Management*, pages 420–425, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-38903-3.

[3] A. Rozinat and Wil M. P. van der Aalst. *Decision mining in business processes*. BETA publicatie : working papers. Technische Universiteit Eindhoven, 2006. ISBN 90-386-0685-0.

[4] Massimiliano de Leoni and Wil M. P. van der Aalst. Data-aware process mining: discovering decisions in processes using alignments. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, page 1454–1461, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450316569. doi: 10.1145/2480362.2480633. URL `https://doi.org/10.1145/2480362.2480633`.

[5] Felix Mannhardt, Sander J. J. Leemans, Christopher T. Schwanen, and Massimiliano de Leoni. Modelling data-aware stochastic processes - discovery and conformance checking. In Luis Gomes and Robert Lorenz, editors, *Application and Theory of Petri Nets and Concurrency*, pages 77–98, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-33620-1.

[6] Wil M. P. van der Aalst. Markings in perpetual free-choice nets are fully characterized by their enabled transitions. In *Application and Theory of Petri Nets and Concurrency: 39th International Conference, PETRI NETS 2018, Bratislava, Slovakia, June 24-29, 2018, Proceedings*, page 315–336, Berlin, Heidelberg, 2018. Springer-Verlag. ISBN 978-3-319-91267-7. doi: 10.1007/978-3-319-91268-4_16. URL `https://doi.org/10.1007/978-3-319-91268-4_16`.

[7] Wil M .P. van der Aalst, Victor Khomenko, Jetty Kleijn, Wojciech Penczek, and Olivier H. Roux. Lucent process models and translucent event logs. *Fundam. Inf.*, 169(1–2):151–177, jan 2019. ISSN 0169-2968. doi: 10.3233/FI-2019-1842. URL `https://doi.org/10.3233/FI-2019-1842`.

[8] Harry H. Beyel and Wil M. P. van der Aalst. Creating translucent event logs to improve process discovery. In Marco Montali, Arik Senderovich, and Matthias Weidlich, editors, *Process Mining Workshops*, pages 435–447, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-27815-0.

[9] Harry H. Beyel and Wil M. P. van der Aalst. Translucent precision: Exploiting enabling information to evaluate the quality of process models. In João Araújo, Jose Luis de la Vara, Maribel Yasmina Santos, and Saïd Assar, editors, *Research Challenges in Information Science*, pages 29–37, Cham, 2024. Springer Nature Switzerland. ISBN 978-3-031-59468-7.

[10] Li Lin, Lijie Wen, and Jianmin Wang. *MM-Pred: A Deep Predictive Model for Multi-attribute Event Sequence*, pages 118–126. 2019. doi: 10.1137/1.9781611975673.14. URL `https://epubs.siam.org/doi/abs/10.1137/1.9781611975673.14`.

[11] B. Gunnarsson, S. Broucke, and J. De Weerdt. A direct data aware lstm neural network architecture for complete remaining trace and runtime prediction. *IEEE Transactions on Services Computing*, 16(04):2330–2342, jul 2023. ISSN 1939-1374. doi: 10.1109/TSC.2023.3245726.

[12] Zaharah A. Bukhsh, Aaqib Saeed, and Remco M. Dijkman. Processtransformer: Predictive business process monitoring with transformer network, 2021. URL `https://arxiv.org/abs/2104.00721`.

[13] Chiara Di Francescomarino and Chiara Ghidini. *Predictive Process Monitoring*, pages 320–346. Springer International Publishing, Cham, 2022. ISBN 978-3-031-08848-3. doi: 10.1007/978-3-031-08848-3_10. URL `https://doi.org/10.1007/978-3-031-08848-3_10`.

Appendices

Appendix A

Some Appendix

Try to avoid appendices. If you can't this is a place to show all kinds of results that are supportive, yet, not critical for your thesis.

Acknowledgments

At first, I would like to express my gratitude to the awesome supervisor that gave me this template.