MASTER

# Quantification of Re-identification Risk in Published Process Models

Maatouk, Karim

*Award date:*
2021

EINDHOVEN UNIVERSITY OF TECHNOLOGY
Department of Mathematics and Computer Science
Process Analytics Group

# Quantification of Re-identification Risk in Published Process Models

*Master Thesis*
*The Erasmus Mundus Joint Master Degree Program in Big Data Management and Analytics (BDMA)*

Karim Maatouk

Supervisors:
Felix Mannhardt(f.mannhardt@tue.nl)

Eindhoven, August 2021

# Abstract

Event logs are the basis of process mining operations such as process discovery, conformance checking, and process optimization. Event logs may contain sensitive information that might be used by adversaries to mount privacy attacks, such as to re-identify individuals that these information belong to, by correlating the event log information with background information the attacker might have. There are multiple techniques to quantify the re-identification risks in published event logs in the process mining community; however, there is no research regarding the quantification of re-identification risk in published process models (Petri nets). Petri nets, which are an output of process discovery and a graphical representation of the process, can also contain sensitive information that can be used to mount privacy attacks. In this thesis, we explore different privacy attacks and show that they are also applicable in the context of a published Petri net. We also propose an approach to quantify the re-identification risk in published Petri nets. Our results show that there is still a significant re-identification risk when publishing a Petri net; however, the results also show the risk is less in published Petri nets than the event logs they were mined from.

# Preface

This Master's thesis titled *Quantification of Re-identification Risk in Published Process Models* has been written to fulfill the graduation requirements of the Erasmus Mundus Joint Master Degree Program in Big Data Management and Analytics (BDMA) and was undertaken at the Mathematics and Computer Science Department at Eindhoven University of Technology (TU/e).

The research questions were formulated jointly with my supervisor, Felix Mannhardt. Firstly, I am extremely grateful to my supervisor for his continuous support and excellent supervision in the duration of this thesis.

Secondly, I would like to thank the BDMA program's coordinators, administrators, and professors for the opportunity of being in this program

Thirdly, I would like to express how grateful I am to my family and friends that motivated me in every step of the way. I would like to extend a special thanks to my friends: Ledia Isaj and Elie Broumana for always being by side and pushing me beyond my limits.

Karim Maatouk
Eindhoven, August 2021

# Contents

# Chapter 1

# Introduction

## 1.1 Thesis Context and Problem Statement

Process mining is the science of understanding processes and improving them based on the mining of event data - process data. Event data is mined for insights that can help industries in optimizing their processes, re-engineering them, and aiding their decision-making. Process discovery is one application of process mining allowing to understand the underlying processes by visualizing the process model.

Process models, which are a graphical representation of the process, can be of different types such as Petri nets, Process tree, Directly-Follows Graphs (DFGs), and others which are further explained in chapter 2. These process models are mined automatically using process discovery algorithms such as Inductive miner [1],[2] [3]. They are mined from event data (event logs), explained in chapter 2. Thus, event data availability is key to any process mining task; without event data, there is no process mining. However, the publishing of event data , in many cases, is subject to constraints due to privacy concerns; hence, limiting the availability of event data. Fields, such as healthcare and other business organizations, make use of process mining techniques for optimizing their processes, and they consider patient information and privacy concerns of utmost importance.

Therefore, researchers in the process mining field aim to find methods to provide privacy guarantees on event logs prior to publishing them. These methods include privacy models to apply anonymization operations on the event logs as discussed in section 2.2.1 to prevent privacy attacks using the event log. These attacks are mainly related to re-identifying information of individuals that was removed from these logs. Researchers, therefore, have worked on quantifying the risk of re-identification of individual information in a published event log to estimate these types of risks such as the works in [4], [5].This allows for an assessment of the effectiveness of a specific anonymization technique on the privacy of an event log.

However, even in cases when the event log is not public, there is still a risk of re-identification of sensitive information using published process models mined from the event log such as Petri nets. Currently, there is still no research addressed to quantify the risk of re-identification attacks based on published Petri nets that are mined using process discovery algorithms from the event log. Therefore, we aim in this work to explore what privacy attacks are possible using the information in a published Petri net. We also aim to quantify the re-identification risk in published Petri nets to provide organizations that publish process models an estimation of the risks posed by doing so. We also aim to provide researchers with a method to evaluate their anonymization methods against the re-identification risk in these published process models. Finally, we aim to assess if it is safer to publish a process model (Petri net) than to publish an event log it was mined from.

## 1.2 Research Questions

Therefore, based on the objectives of this thesis, we identify three key research questions of our work and they are as follows :

1. (RQ1) What are the types of risks associated with publishing a process model (Petri net)?
2. (RQ2) How to quantify the re-identification risk of a process model (Petri net) mined using a process discovery algorithm(Inductive miner) from an event log?
3. (RQ3) Is it safer to publish a process model(Petri net) than to publish the event log it was mined from?

## 1.3 Research Approach

In our research approach, we firstly evaluate the current state of the art regarding privacy-preserving process mining and existing works to quantify re-identification risks in process mining in section 2.2 of chapter 2.

Afterwards, we explore possible privacy attacks that are associated with publishing a Petri net in an attempt to answer our first research question (RQ1). This is done in chapter 3. We explain and illustrate three possible privacy attacks: re-identification, reconstruction, and membership disclosure attacks.

We then move to lay out the theoretical foundation of our approach in 4. In this chapter, we elaborate why the quantification of risk in an un-annotated Petri net is challenging, and we consider only frequency-annotated Petri nets. We, then, detail the challenges in quantifying the re-identification risk and then detail the basic theory of our risk estimation approach of Petri nets mined from event logs using the approach of generating execution traces (or runs) from the Petri net.

Thereafter, we elaborate the implementation of our approach and describe the algorithms used in the implementation while illustrating the subsequent steps of our approach which is done in chapter 5. Therefore, we describe our quantification of re-identification risk approach where we detail reverse engineering of a frequency-annotated Petri net to a simulated event log in two steps: 1)from a Petri net to a process tree, and 2)from a process tree to a simulated event log. Afterwards, we describe how to use existing techniques to quantify the re-identification risk from our simulated event logs to estimate the re-identification risk in a Petri net.

Finally, we describe our evaluation objectives, experimental setup and present the results and discussion of our approach on five real-life publicly available datasets in chapter 6. We also answer our second(RQ2) and third research questions(RQ3)

# Chapter 2

# Background

In this chapter, we start by explaining the preliminaries that are required to understand the context of this work. We then move on to present the state of art in privacy-preserving process mining and the quantification of re-identification risk in the process mining community.

## 2.1 Preliminaries

### 2.1.1 Re-identification Risk

Re-identification risk is the risk of re-identifying individuals in a dataset that had sensitive information about these individuals removed. "Re-identification is the process by which anonymized personal data is matched with its true owner. In order to protect the privacy interests of consumers, personal identifiers, such as name and social security number, are often removed from databases containing sensitive information" [6]. In the process mining field, re-identification risk is mostly researched on anonymized event logs. The trace owners of event logs are usually removed from an event log or anonymized along with their identifying information such as name, birthdates, or addresses. Attackers might use background information they have about individuals and combine their previous knowledge with data from event logs to mount their re-identification attacks. A more elaborate overview of re-identification attacks and other privacy attacks that can occur using event logs or Petri nets is presented in chapter 3. Therefore, re-identification attacks are dangerous, especially in certain domains such as healthcare where patient information is critical.

### 2.1.2 Event Logs

Information systems log huge amounts of events about different processes such as BPM, ERP, and hospital information systems. Events are sequentially recorded in a way that each event corresponds to an activity - defined as a step in the process, and is related to a particular case - defined as an instance of the process. Therefore, it is possible to record a trace of events per case and these cases are considered individual requests.[7] The collection of these traces with additional information about the events is denoted as an Event Log. An example event log of fragment of events of a log to handle requests for compensation is shown in table 2.1.2 [7].

Table 2.1: Fragment of Event Log : Handling Requests for Compensation [7]

| Case id | Event id | Timestamp | Activity | Resource | Cost |
|---|---|---|---|---|---|
| 1 | 35654423 | 30-12-2010:11.02 | register request | Pete | 50 |
| 1 | 35654424 | 31-12-2010:10.06 | examine thoroughly | Sue | 400 |
| 1 | 35654425 | 05-01-2011:15.12 | check ticket | Mike | 100 |
| 1 | 35654426 | 06-01-2011:11.18 | decide | Sara | 200 |

Table 2.1: Fragment of Event Log : Handling Requests for Compensation [7]

| Case id | Event id | Timestamp | Activity | Resource | Cost |
|---|---|---|---|---|---|
| 1 | 35654427 | 07-01-2011:14.24 | reject request | Pete | 200 |
| 2 | 35654483 | 30-12-2010:11.32 | register request | Mike | 50 |
| 2 | 35654485 | 30-12-2010:12.12 | check ticket | Mike | 100 |
| 2 | 35654487 | 30-12-2010:14.16 | examine casually | Pete | 400 |
| 2 | 35654488 | 05-01-2011:11.22 | decide | Sara | 200 |
| 2 | 35654489 | 08-01-2011:12.05 | pay compensation | Ellen | 200 |

Event logs contain in addition to case identifiers such as case id, other event identifiers and event information such as event id, timestamp , activity, resource, and cost. The information in an event log can be used in a variety of process mining operations such as process discovery, process enhancement, and conformance checking.

For each case in an event log, we can write its trace in a concise form representing the activities found in a case. The trace for case 1 in the event log of table 2.1.2 can be represented as :

**Trace:** ⟨register request, examine thoroughly, check ticket, decide, reject request⟩

An event log can also be represented in a compact form as a multiset of traces. An example of a multiset of traces is as follows for the activities a, b, c, d:

**Multiset of traces:** $[\langle a, b, c\rangle^3, \langle a, b, c, d\rangle^2]$

The multiset in this example indicates that the trace ⟨a, b, c⟩ is repeated 3 times in the event log, and the trace ⟨a, b, c, d⟩ is repeated 2 times.

### 2.1.3 Petri Nets

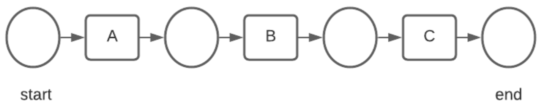
In Process discovery, the output of a discovery algorithm is a Petri net. A Petri net [8] is a graphical representation of the event log. A Petri net system is a type of process models. A Petri net is a bipartite, directed graph consisting of places and transitions. The places of a Petri net, represented as circles, and the transitions as boxes. Arcs are directed from places to transitions or vice-versa.

**Definition 2.1.1** (Petri net)**.** [9] A Petri net is a triple (P, T, F) with:

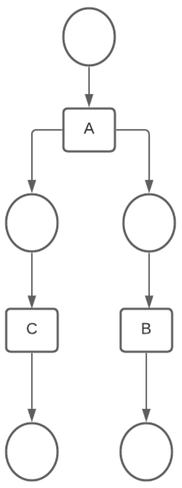
- P is a finite set of places,
- T is a finite set of transitions, and
- F $\subseteq (PT) \cup (TP)$ is a set of flow relations that describe a bipartite graph between places and transitions.

The start and the end nodes represent the source and the sink places of a Petri net. The sequence of transitions in a Petri net represents the activity sequences in an event log. For instance, the Petri net in figure 2.1 illustrates the chronological order that activities occurred in. Activity A occurred, then activity B, then activity C.

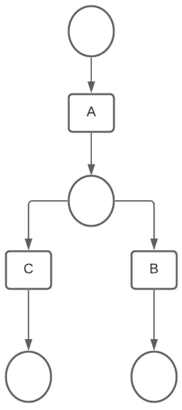
Figure 2.1: Petri Net Basic Example



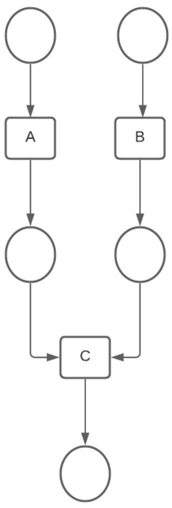
**Workflow Nets (WF-nets):** In this work, we consider a subset of Petri nets known as Workflow nets (WF-nets) and we adopt the definitions in [7]. Workflow nets have a single source place(start) and a single sink place(end). Workflow nets are often used to describe business processes. WF-nets are relevant in business process modeling because in these processes cases, such as insurance claims, have well defined case creation and completion [7]. Therefore, when Petri nets are mentioned in this work, we are referring to their subset class WF-nets.



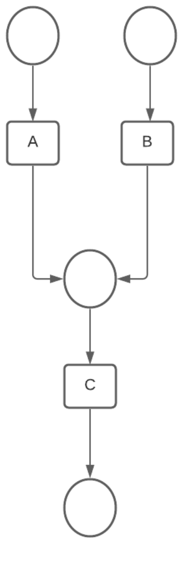
(a) AND-Split



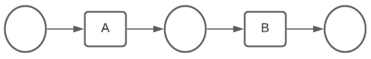
(b) XOR-Split



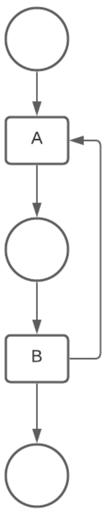
(c) AND-Join



(d) XOR-Join



(e) Sequence



(f) Loop

Figure 2.2: Structural Work Flow Nets Operations

In figure 2.2, we show the structural operations in WorkFlow nets. Petri nets can convey different orders of process activities. Figure 2.2a shows an AND-Split where activity A is followed by both activities B and C. Figure 2.2b shows a XOR-Split where activity A can either be followed by activity B or activity C - representing a choice. Figure 2.2c shows an an AND-join where activity

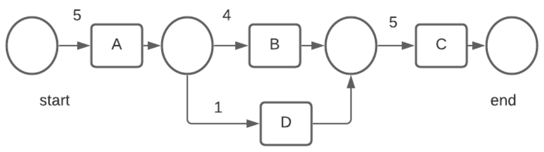
C needs both activities A and B to be able to fire, i.e to happen. Figure 2.2d shows a XOR-Join where activity C can be fired by either activity A or activity B happening. Figure 2.2e shows a simple sequence where activity A fires, then activity B fires. Figure 2.2f shows a XOR loop where activity A and activity B are fired, then it is possible to repeat firing A, B any number of times.

Arcs of a Petri net have the possibility to be labeled with their respective weights that can represent different notations such as :

- Frequency: number of times an arc going into a transition has been fired
- Average time: average time taken to complete a certain transition.

An example of a frequency annotated Petri net can be found in figure 2.3.

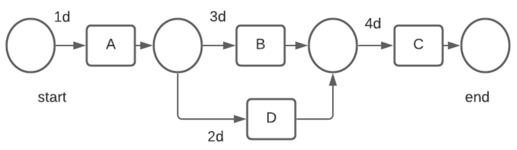
Figure 2.3: Frequency Annotated Petri Net



An example of time annotated Petri net can be found in figure 2.4.

The frequencies on a Petri net can be interpreted as follows : A occurred 5 times, B occurred 4 times, C occurred 5 times, and D occurred 1 time.

Figure 2.4: Time Annotated Petri Net



## 2.1.4 Process Trees

Process trees are another type of process models generated by process discovery algorithms, specifically inductive process discovery techniques [1, 3, 2].Process models using graph-based notations can complicate process discovery and result in unsound process models which complicates discovery. Block-structured models such as process trees are sound by construction [7].

A process tree is defined as follows:

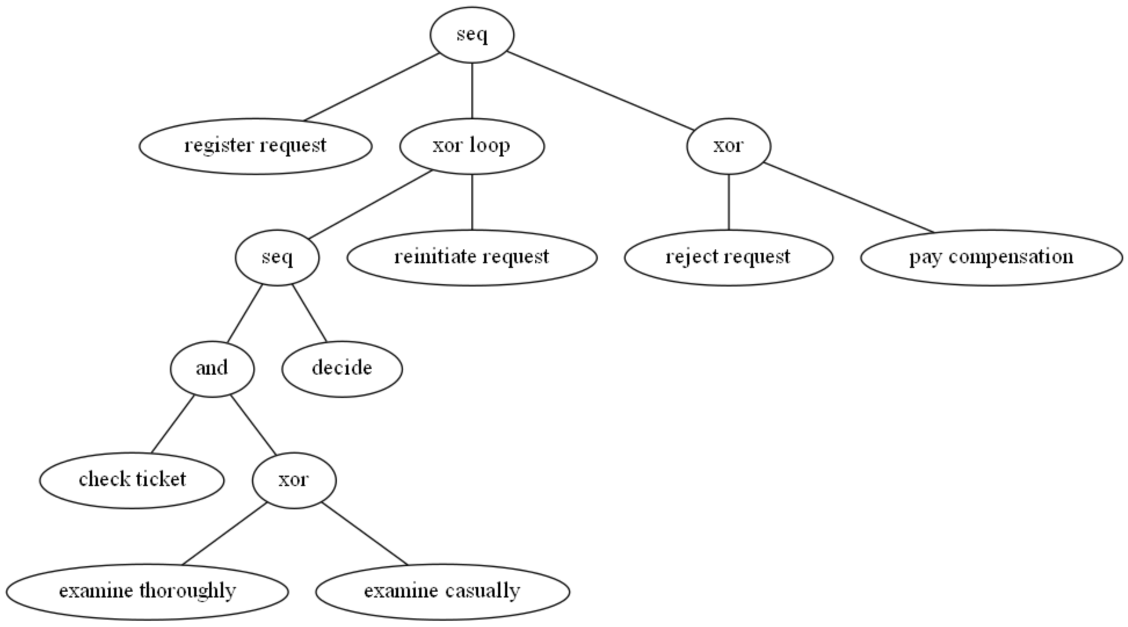
**Definition 2.1.2** (Process Tree)**.** [7] Let $A \subseteq \mathcal{A}$ be a finite set of activities with $\tau \notin A$. $\oplus = \{\rightarrow, \times, \wedge, \circlearrowleft\}$ is the set of process tree operators.

- If $a \in A \cup \{\tau\}$, then Q = a is a process tree,
- If $n \geq 1, Q1, Q2, ..., Qn$ are process trees, and $\oplus \in \{\rightarrow, \times, \wedge\}$, then $Q = \oplus(Q1, Q2, ...Qn)$ is a process tree, and
- If $n \geq 2$ and Q1,Q2,...,Qn are process trees, then $Q = \circlearrowleft (Q1, Q2, ...Qn)$ is a process tree.

$\mathcal{L}_A$ is the set of all process trees over A.

In figure 2.5, a process tree of the event log in section 2.1.2 can be seen.

Figure 2.5: Process Tree of Handling Request For Compensation Event Log [7]



The leaf nodes of a Process tree represent process activities and the other nodes represent operators such as those than can be seen in Petri nets as in fig 2.2 in section 2.1.3. Process trees have four operators: sequence operator , exclusive choice operator, parallel operator, and loop operator denoted as : $\{\rightarrow, \times, \wedge \circlearrowleft\}$ respectively.

The four operators can also be abbreviated as seq, xor, and, xor loop which will be adopted in this work.

The operator nodes define the the order of execution of their children nodes. In the following, we define the semantics of the execution of a process tree by its operator type :

**Definition 2.1.3** (Semantics of a Process Tree)**.** Let $\mathcal{P}$ be a process tree. Let $\mathcal{N} \in \mathcal{P}$ be any non-leaf node. Let $\mathcal{T}(\mathcal{N})$ be of range $\{\rightarrow, \times, \wedge \circlearrowleft\}$ be the type of Node N operator. Let $\mathcal{T}(\mathcal{N})$ have children nodes $\{a, b\}$ with a being the leftmost node, and b being the rightmost node. The execution of the children of the node $\mathcal{T}(\mathcal{N})$ is done as follows:

- if $\mathcal{T}(\mathcal{N}) =\rightarrow$, a is executed then b is executed. Trace is {a,b}
- if $\mathcal{T}(\mathcal{N}) = \times$, a is executed or b is executed. Trace is {a} or {b}
- if $\mathcal{T}(\mathcal{N}) = \wedge$, a and b are executed, a or b may come first. Trace is {a, b} or {b,a}.
- if $\mathcal{T}(\mathcal{N}) =\circlearrowleft$, a is executed, b can be executed any number of times 0...n. For each execution of b, a is executed again. Trace is {a} or {a,b,a} or {a,b,a,...,a,b,a}.

## 2.2 State of the Art

### 2.2.1 Privacy-Preserving Process Mining

Privacy preserving process mining has increasingly gained interest in the process mining community. This comes with legislations and data protection across the EU becoming stricter, especially after General Data Protection Regulation (GDPR)[10]. The main concern of privacy-preserving works in process mining is how to ensure privacy, particularly when publishing event logs that are used in process mining operations such as process discovery, conformance checking, process optimization. Since studied processes can be in fields such as healthcare, financial institutions, and other critical fields, privacy is of utmost importance.

In [11], Mannhardt et al. present possible privacy leakages in event logs and ways to protect against them. They also present a protection model to guard against privacy attacks that makes use of differential privacy concept [12]. In [13], Rafiei et al. introduce a python-based infrastructure that implements privacy-preserving techniques in proceess mining. The work provides an open source tool where privacy-preserving techniques can be integrated. In addition, in [14], Rafiei et al. provide formalization of different anonymization operations that are utilized by privacy models in the process mining field. Three of the most significant privacy models that are applied in process mining are TLKC [15], PRETSA [16], and PRIPEL[17]. In [15], Rafiei et al. provide a k-anonymity mechanism to publish event logs where they present a privacy model TLKC that aims to anonymize event logs before publishing them to provide better privacy guarantees. In [16], Fahrenkrog-Petersen et al. introduce PRETSA algorithm that utilizies k-anonymity and t-closeness to provide privacy guarantees for event logs that are sanitized using the algorithm. The main utility is to avoid the disclosure of identities, memberships and characterization of individuals in an event log based on sensitive attributes. It employs frequency-based filtering to provide privacy guarantees. PRIPEL [17] provides a privacy-aware event log publishing framework. It uses differential privacy mechanism. The PRIPEL framework aims to ensure privacy on the individual cases level in the event log instead of the complete log. Since process mining has several applications in the healthcare domain, and since patient information privacy is of critical importance, there has been also research in the process mining field with regards to privacy-preserving process mining in healthcare, such as in [18] where Pika et al. describe a privacy-preserving framework for process mining in the healthcare field which applies anonymization methods on publicly available healthcare event logs.

### 2.2.2 Re-identification Risk in Process Mining

Quantification of re-identification risk is closely related to privacy-preserving process mining. The quantification of the re-identification risk can help evaluate the effectiveness of privacy-preserving methods in process mining. The risk quantification can act as a comparison measure before and after applying privacy models. Although re-identification attacks are hugely researched in different fields [19], [20], [21], [22], there is only a couple of published research in the process mining community on quantification of disclosure risk in event logs and directly-follows graphs.

The most related to our research are found in [4], [5], and [23]. In [4], Nunez von Voigt et al. present a method to quantify the re-identification risk in event logs. The authors propose two measures to quantify the risk. Both measures are based on uniqueness in the event logs. The two measures are: uniqueness based on case attributes and uniqueness based on traces. In the first measure, the uniqueness of the case attributes in an event log is used to estimate the re-identification risk. The second measure considers the uniqueness of traces in an event log to account for event logs that do not have a lot of case attributes where the only information in the event log is the traces themselves. The work quantifies the risk in publicly available event logs and demonstrates how the re-identification risk can be very high for some of them and that almost every case can be re-identified in some scenarios. This sheds light on the need for adequate methods to quantify the re-identification risks and means to protect against them.

In[5], Rafiei et al. introduces two measures for quantifying disclosure risk in published event logs to evaluate the effectiveness of privacy-preserving techniques. The two proposed measures are identity(case) disclosure and attribute(trace) disclosure. The first measure, case disclosure, uses uniqueness to measure how trace owners can be re-identified. The second measure, trace disclosure, measures how the sensitive attributes such as the complete trace of a case, can be disclosed. The method takes into account the background knowledge that the attacker might posses about the event log into account when quantifying the risk. The method considers three types of background knowledge: set, multiset, and sequence. The set background knowledge is simply the set of activities in a process that the attacker might know are related to an individual. A multiset background knowledge provides additional knowledge about the number of occurrences of the process activities, while the sequence background knowledge provides addiional information about the order in which the activities have occurred in for the trace owner. The paper applies the method on two publicly available real-life event logs.

In [23], Elkoumy et al. discuss the re-identification probability in directly-follows graphs (DFGs), which are an output of process mining techniques. The work expresses the re-identification and the guessing advantage of an attacker by calculating the guessing probability given a DFG.

The previous mentioned works focus on the disclosure risk of an event log and Directly-follows graphs (DFGs) in process mining; however, there is not any work on the quantification of re-identification risk in published process models such as Petri nets or process trees. Therefore, the importance of this work comes from the lack of any work in that area and the importance of the quantification of re-identification risk in published process models. Since a lot of organizations publish process models within the organization or publicly, it is important to know what the disclosure risk and the consequences of publishing such models are.

Chapter 3

# Petri Nets and Associated Privacy Risks

In this chapter, we examine privacy attacks that are possible given a published Petri net and attempt to answer our first research question (RQ1).

One output of process discovery algorithms is Petri nets. Petri nets are a graphical representation of a given process that can help understand the underlying process in an abstract manner. Petri nets are becoming more and more used in different domains such as business and healthcare. Organizations publish event logs and process models to stakeholders or to the public for documentation, process optimization, or process analytics.

Since event logs and process models contain information, there are risks of privacy attacks that seek to reveal sensitive information given the attacker has some background information about the individuals in an event log or process model.

Previous works addressed the risks associated with publishing event logs [4], [5].Other works also include possible attacks related to publishing of DFGs (Directly Follows Graphs) [23]. However, no assessment of risks and attacks based on published Petri nets is available in the process mining research community.

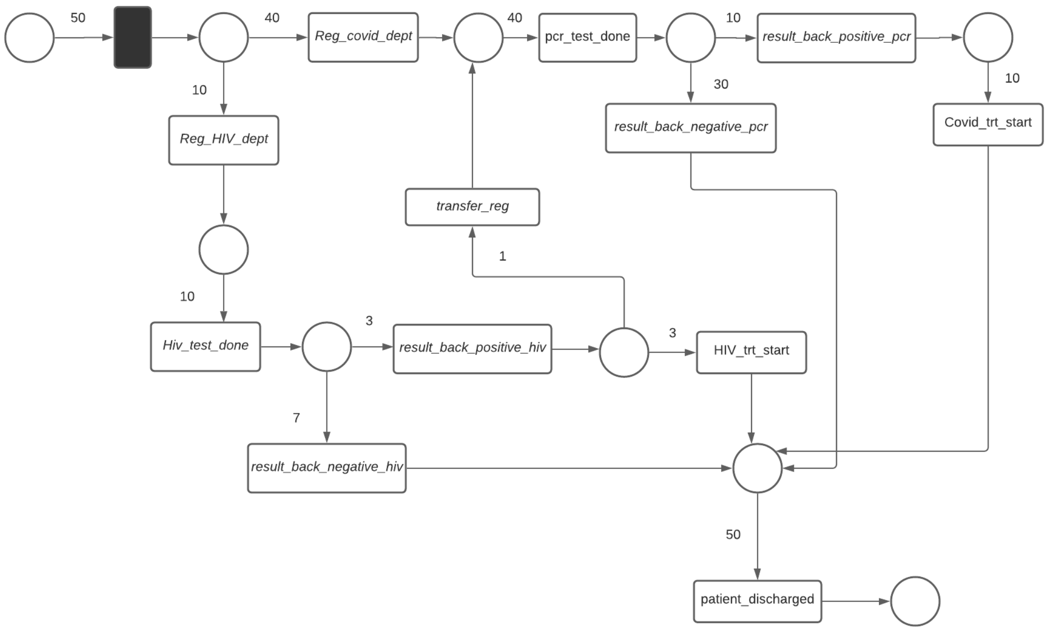
To address this, we discuss the feasibility of common privacy attacks in the context of published process models - Petri nets; that is : Re-identification , Re-construction, and Membership disclosure attacks.

Petri nets can include information about certain actors or resources, in addition to trace information. Therefore, Petri nets can express plenty of information about a certain process, organization , or resources. This is why it is important to assess whether the published Petri net can reveal more than intended to stakeholders or to the public that might constitute a basis for a privacy attack.

## 3.1 Privacy Attacks in Petri nets' Context

Petri nets can represent a variety of information that might unintentionally be revealed by the publisher of the Petri net to stakeholders or to the public. In other scenarios, a privacy attack might occur and an adversary might be able to disclose sensitive or classified information based on the published Petri net. Although the data might not be present explicitly in the Petri net, the attacker might be able to draw out sensitive information by using different techniques; especially when combined with other data obtained from a breach or other publicly available data about the individuals. In this section we explore different attacks that an adversary might launch and establish how they can do so using a Petri net in an attempt to answer our first research question(RQ1). For the purposes of illustration, we use the following example in figure 3.1 of a testing process for COVID and HIV in a medical facility. The process illustrates the process from the point of registration in the relevant department (COVID or HIV) up to the testing process, result (negative or positive), and the discharging of the patient. The Petri net is frequency-annotated with the number of occurrences of the respective activities (transitions).

Figure 3.1: Medical Center COVID, HIV testing process April 2, 2020



### 3.1.1 Re-identification Attacks

A re-identification attack occurs when an adversary attempts to reverse the anonymity of certain information that was masked by an operation to remove identifying or sensitive information. The adversary can use information that was acquired publicly or during a data breach. Attacks are most successful when adversaries correlate or match different datasets and information to mount an attack.

Petri nets may contain sensitive information about individuals or processes. The adversary might attempt to use the information in the Petri nets to re-identify individuals in their attack.

The Re-identification risk has been studied in the literature on different types of data. In [4], the re-identification risk is assessed on event logs , and in [23] the disclosure risk is assessed on Directly Follows Graphs. However, the re-identification risk has not been assessed in published Petri nets. In a dataset or an event log, an adversary will attempt to single out an individual's identity based on the uniqueness of a record's or event's identifiers in order to mount a re-identification attack. Singling out a record, the adversary can attempt a linkage attack using other datasets obtained by the adversary which can lead to the re-identification of individual information.

In table 3.1.1 below we give an example of fragment of an event log for the Petri net in figure 3.1 for COVID/HIV testing process. We also demonstrate an example of the background knowledge that an attacker might posses about the process in table 3.1.1. The event log has its sensitive information about individuals such as name, birth date, sex removed from the event log.

Table 3.1: Example Event Log COVID, HIV Testing Process in a Hospital

| Case id | Event id | Timestamp | Activity |
|---|---|---|---|
| 1 | 1000 | 02-04-2021:08.02 | reg_covid_dept |
| 2 | 1001 | 02-04-2021:09.05 | reg_HIV_dept |
| 3 | 1002 | 02-04-2021:11.10 | transfer_reg |
| 4 | 1003 | 02-04-2021:13.20 | reg_covid_dept |
| 6 | 1004 | 02-04-2021:15.30 | result_back_negative_pcr |
| 7 | 1005 | 02-04-2021:17.30 | pcr_test_done |
| 3 | 1006 | 02-04-2021:18.00 | result_back_positive_HIV |
| 9 | 1007 | 02-04-2021:18.30 | reg_covid_dept |

In the event log above, we notice a unique activity transfer_reg (Event 1002) with a timestamp on April 2,2020. Although the information in the event log alone does not reveal any information about the individual - the trace owner; however, when correlating the event log information with background knowledge that the attacker might have about the process, the attacker is able to identify the individual that performed the activity to transfer departments transfer_reg which is Carla Sanders based on the uniqueness of date April 2,2020 and the activity transfer_reg. The attacker, thus, is able to identify the individual's name by correlating the event log information with the background knowledge available about the process. Not only that, but the attacker is also able to know the complete activities (trace) of Carla Sanders based on the case identifier. Therefore, the attacker also knows that Carla Sanders has tested positive for HIV by the event result_back_positive_HIV in the event log which has same case id 3.

Table 3.2: Attacker Background Knowledge: COVID Department Registrations

| Patient Name | Registration | Date |
|---|---|---|
| John Smith | reg_covid_dept | April 2,2021 |
| Carla Sanders | transfer_reg | April 2,2021 |
| Sara Danvers | reg_covid_dept | April 2,2021 |
| John Legend | reg_covid_dept | April 2,2021 |

In the context of a frequency-annotated Petri net, a similar paradigm can be established to understand how an adversary can use the Petri net information to re-identify individuals.

Singling out individuals in a Petri net can be done based on infrequent paths(runs) in Petri net. An infrequent path in a Petri net allows an adversary to single out an activity of that process. To illustrate, in figure 3.1, we notice a single case in which a patient transferred from the HIV testing department to a COVID testing department. This information does not reveal the individual's identity; however, coupled with other information that the attacker might have, it can lead to a successful re-identification by an adversary. For instance, if the adversary has background information about patients registrations for COVID department, such as in table 3.1.1, and they detect that a unique individual transferred from HIV department, without undergoing registration directly for a PCR test, they can conclude that that individual is Carla Sanders from the background information. THey can also know from the Petri net that Carla Sanders has also undergone an HIV test as well as the result of the HIV test( positive). Therefore, the individual identity and HIV results are revealed in that attack. In the Petri net in figure 3.1, there is a single activity transfer_reg after ongoing a result_back_positive_hiv as indicated by the frequency on the activity(transition). Also in the background information in 3.1.1, Carla Sanders transferred registration to COVID department; Therefore, the adversary can identify Carla Sanders as the individual belonging to the trace containing transfer_reg and also can know the complete trace prior to the transfer, such as that she had undergone a positive HIV test result_back_positive_HIV.

By singling out the infrequent trace transfer_reg of frequency 1 of the Petri net in figure 3.1, the adversary mounts a linkage attack using background knowledge in table 3.1.1 to re-identify the individual as Carla Sanders who is the only one that has undergone transfer on that day April 2, 2020; and thus can identify that she has undergone a positive HIV test.

Therefore, we show that a re-identification attack is also possible using the information available in a published Petri net.

### 3.1.2 Re-construction Attacks

A reconstruction attack occurs when an adversary attempts to reconstruct the original dataset from aggregated data or anonymized one. The re-construction can be partial or complete. The adversary might use other available data to mount a more successful reconstruction attack. In the context of a published Petri net, a re-construction attack means the attacker might aim to reconstruct the original event log which leads to a privacy violation.

One example is singling out an individual due to infrequent paths in a Petri net which can lead to inferring information about the original event log. From the previous example in figure 3.1 and background knowledge in table 3.1.1, a successful re-identification can be extended to a reconstruction attack. The re-identification of Carla Sanders by matching with other available background information can lead to obtaining different information about the individual such as age, sex, occupation, and other attributes from the event log and the exact events in an event log. For instance, we give an example of other information that an adversary might be able to obtain from a voter registration list in table 3.1.2.

Table 3.3: Attacker Background Knowledge: Voter Registration

| **Voter Name** | **Zip code** | **Sex** | **Birth date** |
|---|---|---|---|
| John Smith | 5622 | male | 15/02/1998 |
| Carla Sanders | 5611 | female | 12/11/1964 |
| Sara Danvers | 1101 | female | 10/05/1958 |
| John Legend | 3303 | male | 13/09/1988 |

Using this information above, along with the information from the background knowledge in 3.1.1 and the trace information from the Petri net in figure 3.1, an attacker might be able to construct an event log with its original sensitive information such as name, age, sex, zip code, and trace information about these individuals activities in the hospital COVID/HIV testing process which results in a serious privacy violation. In this case it would be a partial reconstruction. However, in more sophisticated attacks, the linkage of many more datasets can happen leading to a more complete event log reconstruction from a published Petri net model.

Therefore, we demonstrate the feasibility of reconstruction attacks using a published Petri while correlating background knowledge and other datasets.

### 3.1.3 Membership disclosure Attacks

A Membership disclosure attack occurs when an adversary attempts to establish whether a certain individual is involved in a given dataset. In this scenario, we consider whether an individual participated in a certain activity in the published Petri net as the predicate of membership . This attack can be mounted in different ways. A linkage attack can also have guaranteed success in case of infrequent traces in a Petri net. In the example Petri net of figure 3.1 and background knowledge in table 3.1.1 , if an adversary wants to know whether Carla Sanders participated in the activity HIV test, they can correlate the data to figure out that she was the single case who transferred departments on April 2, 2020. Thus, figuring out her membership in that activity. The adversary can also resort to different techniques to infer the participation of an individual in the process as a whole or a certain activity. Therefore, we demonstrate also the feasibility of membership disclosure attacks in the context of a published Petri net.

# Chapter 4

# Re-identification Risk Quantification in Petri Nets

In this chapter, we explain the challenges of quantifying the re-identification risks from Petri nets. We also discuss why a brute-force approach to compute the re-identification risk is computationally expensive, and we motivate for an estimation approach to quantify the re-identification risk in frequency-annotated Petri nets.

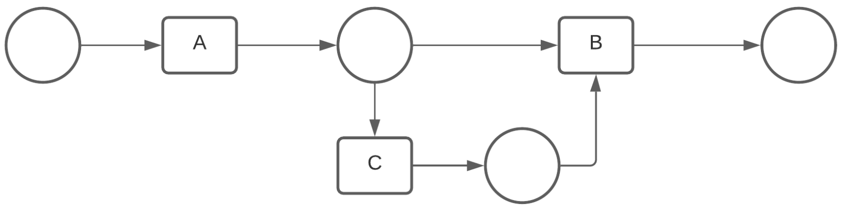
## 4.1 Un-annotated Petri Nets : Quantification of Risk

In this section, we analyze why the quantification of risk in un-annotated Petri nets is challenging, and could rather be not feasible. Un-annotated Petri nets are Petri nets that do not present information about the frequency of the transitions(activities)in the net.

In order to estimate the risk of publishing any kind of information, one needs to analyze what kind of information is revealed that can pose risk of re-identifying records from the information. Similarly, we need to analyze the kind of information that can be revealed by an un-annottated Petri net.

Consider the simple Un-annotated Petri net in figure 4.1 consisting of activities A, B, and C.

Figure 4.1: Un-annotated Petri net Example



Information that can be revealed by this un-annotated Petri net can be :

- Process activities order : The Petri net in figure 4.1 reveals that activity A is either followed by activity C or activity B. Activity C is followed always by activity B. A is the start activity and B is the end activity
- Possible process traces: The Petri net in figure 4.1 reveals the possible traces of the Petri net which can be either $\langle A, B\rangle$ or $\langle A, C, B\rangle$

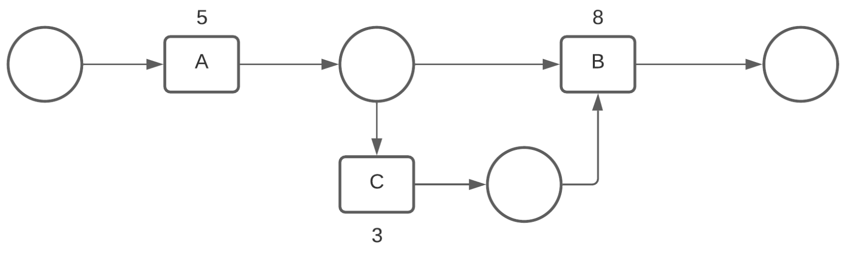
While the information that is revealed by an un-annotated Petri net is mostly related to Process control-flow, and in some cases might pose a privacy threat to organizations' operation, quantifying the re-identification risk is challenging. This is mainly because most methods of estimation of the re-identification risk rely on the uniqueness measure of records or traces. However, this information is not revealed in un-annotated Petri nets. On the contrary frequency-annotated Petri nets reveal such information as they decorate activities with their respective frequencies. This both reveals more information about possible traces in the Petri net and makes the estimation more possible with that kind of information.

## 4.2 Generation of Execution Traces from a Petri Net

Prior to quantification of any kind of risk that might be incurred by publishing a Petri net, we need to assess what kind of information can be revealed by a Petri net. For that, we demonstrate that it is possible to gain some information from a Petri net such as the corresponding traces of the Petri net.

In figure 4.2, we show an example of a frequency-annotated Petri net.

Figure 4.2: Frequency-annotated Petri net



It is possible to generate from this Petri net its corresponding execution traces as follows:

- $\langle A, B\rangle^5$
- $\langle A, C, B\rangle^3$

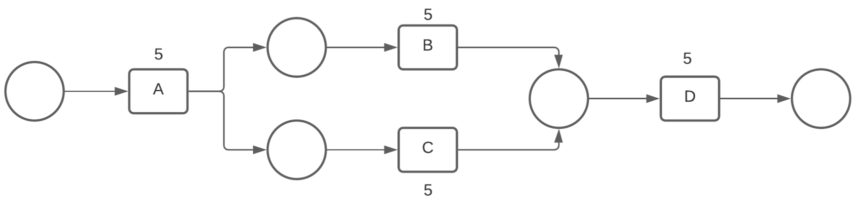
However, it is not always possible to generate the corresponding traces from a Petri net particularly because of the non-deterministic nature of some Petri nets as addressed in the challenges below.

### 4.2.1 Challenge 1: Petri Nets Can Have Concurrent Transitions

Petri nets can express multiple relationship types between process activities in its semantics. One of the relationships expresses concurrent or parallel behaviour as can be seen in figure 2.2 explaining structural WorkFlow nets operations in section 2.1.3 where parallel behavior is expressed in the AND-Split and AND-Join of the Petri net.

In figure 4.3, we demonstrate an example frequency-annotated Petri net showing concurrent transitions in a Petri net.

Figure 4.3: Concurrent Transitions in a Petri net



The transitions B and C in the Petri net express concurrent behaviour meaning that either transition B can be fired first, then transition C; or transition C can be fired first, then transition B.

This concurrency constitutes a challenge when generating the corresponding execution traces of a frequency-annotated Petri net.Altough the exact number of transitions is known in the Petri net, multiple options are present when generating the traces.

In the Petri net of figure 4.3, there are 6 possible traces that can satisfy the frequencies of the transitions in the Petri net and they are as follows:

1. $\langle A, B, C, D\rangle^5$
2. $\langle A, C, B, D\rangle^5$
3. $\langle A, B, C, D\rangle^4$, $\langle A, C, B, D\rangle^1$
4. $\langle A, B, C, D\rangle^1$, $\langle A, C, B, D\rangle^4$
5. $\langle A, B, C, D\rangle^3$, $\langle A, C, B, D\rangle^2$
6. $\langle A, B, C, D\rangle^2$, $\langle A, C, B, D\rangle^3$

Thus, this constitutes a challenge to know which exact traces correspond to the original event log of which a Petri net was mined from. For this transition, there is a 1/6 chance of generating the exact traces corresponding to the original event log which the Petri net was mined from. This chance decreases when nesting multiple transitions; thus, decreasing the chance of identifying the original traces of the log.

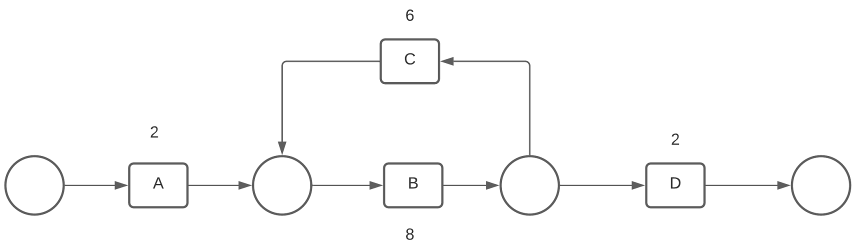
Another type of semantics in an event log that constitute a challenge for identifying the original traces of the Petri net is the Loop or cyclic relationship as addressed in the next section.

### 4.2.2 Challenge 2: Petri Nets Can Be Cyclic

Another type of relationship semantics between process activities in a Petri net is a cyclic or loop relationship. This property results in Petri nets being deterministic in certain cases. This also constitutes a challenge for generating the execution traces corresponding to the original event log, of which the Petri net was mined from.

In figure 4.4 , we demonstrate an example of a Cyclic Petri net. Transition C is an optional loop that acts as a redo transition for the transition B of the Petri net.

Figure 4.4: Cyclic Frequency-annotated Petri Net



Although in figure 4.4 the frequencies of the transitions of the Petri net are known, the exact traces corresponding to the original log which the Petri net was mined from cannot be known exactly as there multiple possible set of traces corresponding to the Petri net.

The are 3 possible multiset of traces corresponding to the Petri net in figure 4.4 and they are as follows:

1. $\langle A, B^4, C^3, D\rangle$, $\langle A, B^4, C^3, D\rangle$
2. or $\langle A, B^5, C^4, D\rangle$, $\langle A, B^3, C^2, D\rangle$
3. or $\langle A, B^6, C^5, D\rangle$, $\langle A, B^2, C, D\rangle$

The number of possible varied multisets of traces can increase when the frequency of the loop or the number of total traces of the Petri net increase. The possible traces' mutlsiets can also increase when there is nesting of multiple cyclic or concurrent transitions in the Petri net.

Therefore, this also constitutes a challenge when generating the exact execution traces that correspond to the original log of the Petri net.

## 4.3 Quantification of Re-identification Risk from Traces of Petri Nets: An Estimation Approach

As mentioned in section 2.2, existing re-identification risk quantification techniques calculate re-identification risk from event logs which constitute of traces.

The idea to for calculate the re-identification risk from frequency-annotated Petri nets is to generate the traces corresponding to the original event log from the Petri net.

The generation of the exact traces corresponding to the original log from the frequency-annotated Petri net is not possible in all Petri nets. This can be done accurately for Petri nets that do not contain concurrent transitions and cycles as described in the challenges described earlier. This is due to the fact that when the Petri net contains concurrent transitions and cyclic behaviour, there can be multiple executions traces, of which we do not know which correspond to the original event log.

The brute force approach to calculate the re-identification risk would consider the generation of all the combinations of transitions in a Petri net and their possible traces sets which could be very computationally expensive. Afterwards, the average risk would be calculated from sets of all traces.

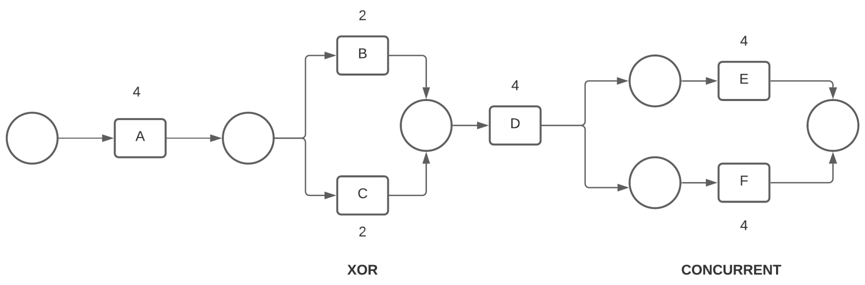
However, since most Petri nets that describe business processes in fact contain concurrent and cyclic behaviour, we need to adapt the solution to estimate the quantification risk using approximation techniques.

Our proposed approach is to generate execution traces from the frequency-annotated Petri without considering all the possible traces sets in a Petri net. This is because when the Petri net has many transitions and especially nested cyclic and concurrent ones, the number of possible traces sets increases substantially. Afterwards, the quantification of risk can be done on the approximated possible traces of the Petri net.

Since the brute-force approach is not computationally feasible, we try to approximate the solution in our approach which is detailed in section 5. The idea is to generate the execution traces in an ordered manner and not consider all the possible traces combinations. This can be done by traversing the Petri net in an ordered manner without randomizing the options to fire transitions, Therefore, for XOR transitions and concurrent transitions, we only consider a fixed order of firing of the activities, and not all the combinations possible.

We give an example in figure 4.5 of a Petri net containing XOR and concurrent transitions.

Figure 4.5: Petri Net with Concurrent and XOR Transitions



The estimation approach proposed would traverse the Petri net above in an ordered manner, in the following order :

1. A is fired
2. When XOR transitions B and C are reached, B is fired first, then C is fired.
3. D is fired
4. When Concurrent transitions E and F are reached, E is fired first, then F is fired.
5. Multiple traversals are executed until the frequencies on the transitions are satisfied

The brute force approach would have considered all the combinations of firing B first or C first in a XOR transitions as well as E first, then F first in concurrent transitions. This would have resulted in many combinations that are possible for the traces.

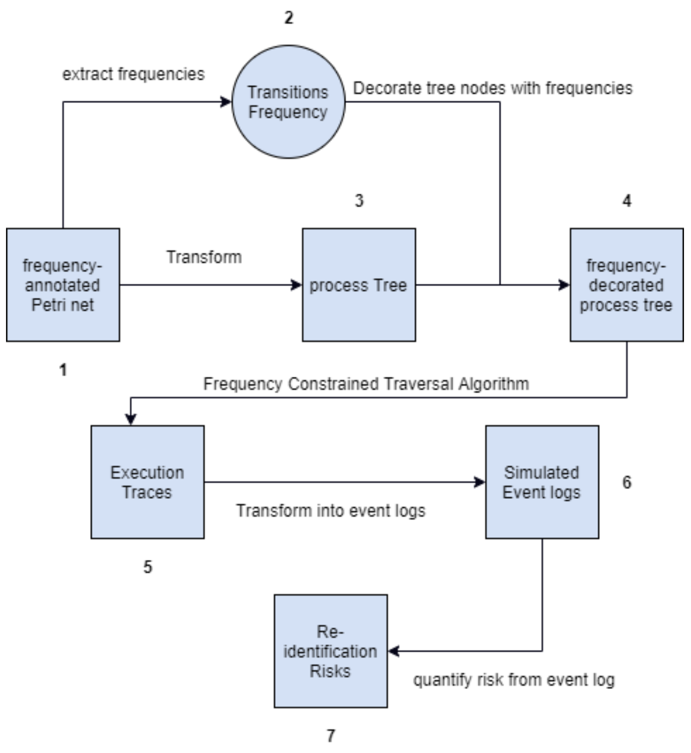
Therefore, the approximation we propose considers only the ordered firing of the transitions in a Petri net to generate the execution traces which results in less combinations of the transitions. The approach chapter 5 explains the application of the method in details for generating the traces and applying existing methods to quantify the re-identification risk. The evaluation of the approach is carried out in the evaluation section 6.3 to assess whether the proposed method could be a good estimator of the re-identification risk.

# Chapter 5

# Approach

In this chapter, we describe the approach we implemented to quantify the re-identification risk from published process models - namely Petri nets. The chapter includes details of theoretical and technical implementations behind the followed approach outlining any assumptions. In the previous chapter, we explained the challenges in quantifying the re-identification risk from Petri nets and the how the execution of the runs of a frequency-annotated Petri net system can help estimate the re-identification risk from the generated runs of the system. Therefore, we suggest an approach to generate the runs of a Petri net system and calculate the re-identification risk from the runs using existing methods that estimate the risk from event logs. Section 5.1 discusses the reverse-engineering of a frequency annotated Petri net to a simulated event log. Then, section 5.2 discusses the application of an existing re-identification risk quantification method on the simulated event log. An overview of our approach is shown in figure 5.1.

Figure 5.1: General Overview of Approach


2
extract frequencies
Transitions Frequency
Decorate tree nodes with frequencies
3
4
frequency-annotated Petri net
Transform
process Tree
frequency-decorated process tree
1
Frequency Constrained Traversal Algorithm
Execution Traces
Simulated Event logs
6
Transform into event logs
5
Re-identification Risks
quantify risk from event log
7

## 5.1 Reverse Engineering a Frequency Annotated Petri Net

In this section, we describe the approach taken to generate a simulated event log from a frequency-annotated Petri net in a manner satisfying the frequencies on the transitions of the system. We first detail transforming a Petri net into a corresponding process tree then generating an event log from the process tree according to the Petri net transitions frequencies.

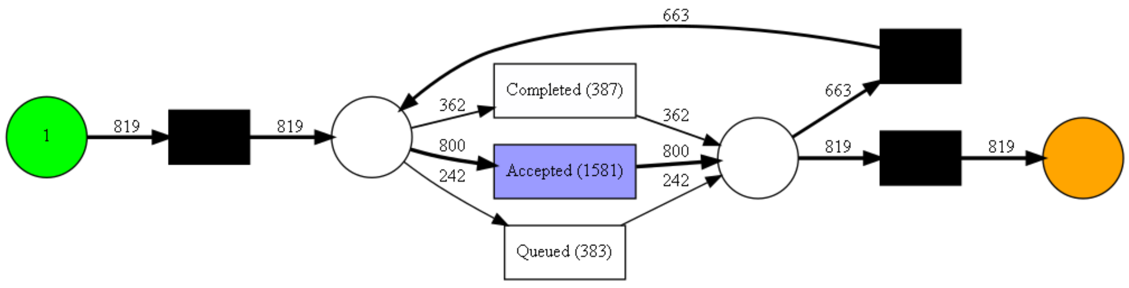
### 5.1.1 From a Petri net to a process tree

In this section, we detail the approach to transform a Petri net system to its corresponding process tree. Firstly, we describe the motivation of the transformation. Afterwards, we describe the method used. The Approach applied considers only a subset of Petri nets which is Workflow nets.

The motivation behind transforming a Petri net to a Process tree is that a process tree enables a more direct expression of the control flow operations of a Petri net. The semantics of a process tree allow for a more straightforward expression of the control flow. The traversal of the process tree nodes enables the generation of traces that are allowed by the Petri net.

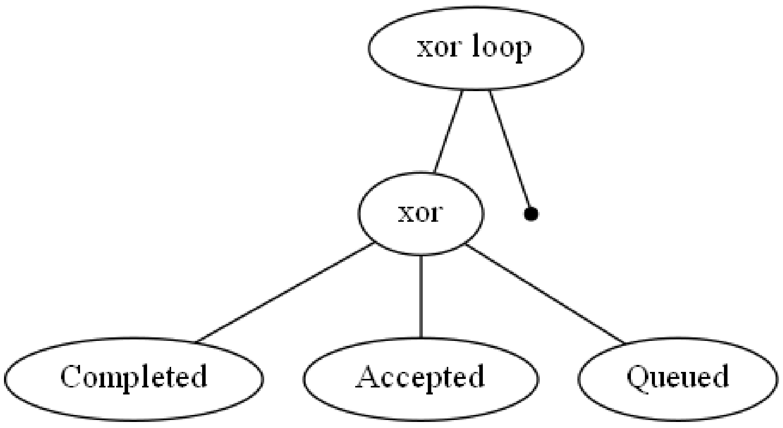
In figure 5.2, we observe a simple frequency-annotated Petri net of the BPIC 2013 Open Problem event log [24]. The Petri net contains 3 activities : Completed, Accepted, and Queued.

Figure 5.2: Petri net of BPIC_2013_Open_Problem



The application of the algorithm described by van Zelst in [25] , allows the transformation of a sound Workflow net into its corresponding process tree as can be seen in Figure 5.3. The transformation into a process tree preserves the control flow of the Workflow net and its process activities. The basic operations of a Petri net such as Parallelism, XOR, Loop, and Sequence are equally expressed in a Process tree which allows the same traces to be played by a process tree.

Figure 5.3: Process Tree of BPIC_2013_Open_Problem



In the process of transforming a Workflow net into its corresponding Process tree, we aim to preserve the frequencies of the transitions of a Petri net in the transformed process tree. We do that for two types of transitions : silent transitions (taus) and normal process transitions (activities).

#### Process Transitions

The process of mapping the frequencies of Petri net transitions into those of a Process tree is straightforward as all the activities names of a Petri net are kept intact such as Completed, Accepted, and Queued in Figure 5.3. The mapping is done by a simple algorithm to map the names and frequencies of the transitions from the annotated Petri net to the Process tree.

#### Silent Transitions

On the other hand, silent transitions need additional handling to infer their respective frequencies. This is due to the fact that some silent transitions are reduced in the process of transforming the Petri net into a Process tree as some silent transitions are deemed as useless and their identifiers removed in the process mining library Pm4py implementation. For instance, in Figure 5.2, the Petri net had 3 silent transitions, represented in black, while, in Figure 5.3, the corresponding process tree had only one silent transition intact, represented in a black dot. Another challenge is that the naming of the silent transitions is removed in the process of creation of a process tree. A workaround that is implemented is to preserve the naming of silent transition in the structure of a Process tree which allows to apply a straightforward mapping of the frequencies from a Petri net by their names. This is done by adding an extra field in the process tree structure in Pm4py implementation to preserve the naming of the silent transitions.

After the Workflow net is transformed into a Process tree while preserving the frequency information, the next step is to generate the execution traces from the obtained Process tree as explained in the next section.

### 5.1.2 From a process tree to an event log

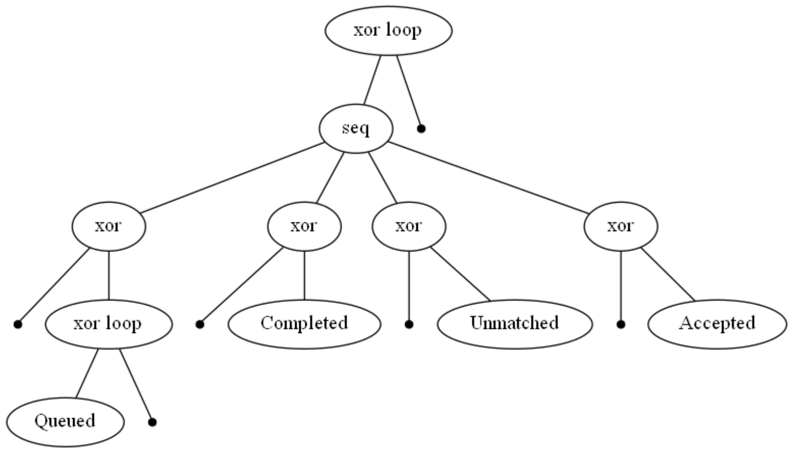
In this section, we explain the generation of a simulated event log from a process tree according to the frequencies of the transitions obtained from a Petri net. Below, we list the different steps taken to generate a simulated event log from a process tree. The first step is to assign the frequencies of the leaf nodes from the frequency annotated Petri net to the process tree nodes as it does not have the frequency information. Leaf nodes are the transitions of a Petri net including silent transitions. The second step is to infer frequencies of all other nodes of the process tree. The third step is to traverse the process tree using the algorithm implemented in our research to generate the execution traces from the process trees. The simulated traces are then transformed into a simulated event log.

#### Step 1: Leaf Nodes Frequencies

In the first step, we decorate the leaf nodes of the process tree with their respective frequencies that are mined from the Petri net. The leaf nodes of a process tree, which represent the transitions of a Petri net, are assigned their frequencies. The reason this is done is to simulate the process tree according to the frequency constraints of the transitions mined from the Petri net. This will allow us to determine the desired number of times each transition can be fired in our simulation of the process tree.

In Figure 5.4 below, the process tree of the BPIC 2013 Incident log [24] is used as an example. The process of decoration of the leaf nodes with their respective frequencies is done. The process tree is traversed to assign frequencies to each leaf node.

Figure 5.4: Process tree of BPIC_2013_Incident Log[24]



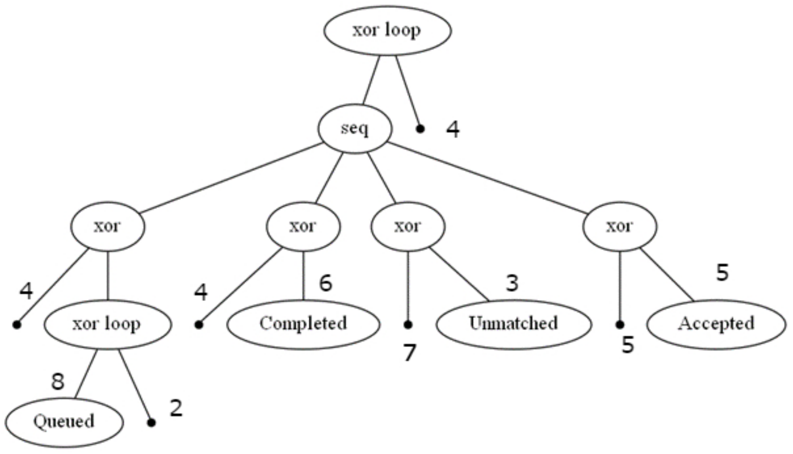
The traversal algorithm, given an input dictionary with the frequencies of a Petri net transitions, traverses the tree starting from the root node, checks if the node is a leaf node, and assigns the frequency of the transition from the input dictionary to the matched leaf node. The algorithm continues to check the children of each node applying the same logic.

**Algorithm 1:** Algorithm for assigning leaf nodes frequencies

```
Input: Process tree PT , Petri net transitions frequency dictionary Dict_Petri
Output: Dictionary Process tree leaf nodes with frequency count
1  Stack ← [PT] // Add tree to stack
2  Dict_tree ← Initialize
3  while length(Stack) > 0 do
4      Node = Stack.pop(0) ;
5      for child in Node.children do
6          if child isLeafNode(): then
               // Assign frequency of petri net transition to process tree node
7              Dict_tree[child] = Dict_Petri[child.name] ;
8          else if child.hasChildren() then
               // Traverse children of the node
9              Stack.append(child) ;
10     end
11 end
```

The result of the algorithm will be a dictionary with the process tree elements and their respective counts. To illustrate this visually, we use the example below in Figure 5.5 where the leaf nodes of the process tree of BPIC_2013_Incident log are assigned their respective frequencies.

Figure 5.5: Assigned Leaf Nodes Frequencies Process tree of BPIC_2013_Incident Log[24]



In order to properly simulate the process tree into its respective execution traces, we need to assign each node of the process tree with its respective desired frequency. This is done in the next step where we infer the frequencies of each node in the process tree prior to its frequency-constrained traversal to generate the simulated traces.

**Step 2: Infer Frequencies of all Process Tree's Nodes**

In the second step of the approach, we decorate the rest of the process tree nodes by inferring the frequencies of these nodes from their children's leaf nodes. According to the type of the nodes of the process tree, a formula is applied to calculate the frequency of the parent nodes from their children nodes. The assignment of the frequencies to all the nodes of a process tree will allow the simulation of the execution traces from the process tree according to the frequencies of these nodes. The execution traces are then transformed into their corresponding simulated event log.

In figure 5.5 above, we can see the leaf nodes decorated by their frequencies. To extend the frequencies to the rest of the node, we apply the following formula according to the type of the node's operator:

**Definition 5.1.1** (Frequency of Node)**.** Let P be a process tree, T(N) be the type of the node operator at a node N which can be ∈: $\langle SEQ, XOR, AND, LOOP\rangle$.

Let N be a node of a process tree which can be equal to P or any subtree of P that is a non-leaf node. The frequency F of node N is defined as :

$$F(N) = \begin{cases} F(first(N.children)), & \text{if T(N) = SEQ} \\ F(first(N.children)), & \text{if T(N)= AND} \\ \text{F(Left(N.children)) - F(Right(N.children))}, & \text{if T(N) = LOOP} \\ \sum_{child=left(N.children)}^{right(N.children)} F(child) & \\ & \\ & \text{if T(N) = XOR} \end{cases}$$

Where *first* returns the first child of a node, *Left* returns the leftmost child of a node, and *Right* returns the rightmost child of a node. The calculation of the frequencies of the nodes is possible recursively from the node being calculated to its children nodes. The function F(N) traverses the process tree in a top-bottom approach; however, the assignment of the frequencies id done from the bottom-up recursively.

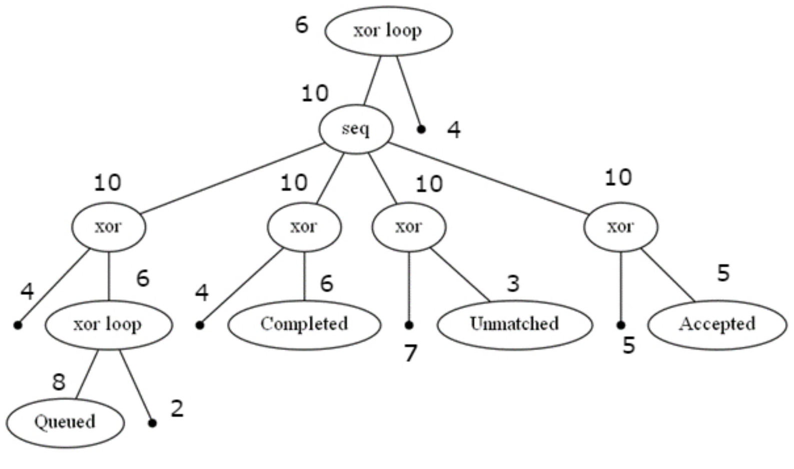
As per definition 5.1.1, we implement the following recursive algorithm to traverse the process tree, and infer the frequencies of each of the non-leaf nodes of the tree:

**Algorithm 2:** Recursive Algorithm for inferring frequencies of non-leaf nodes

```
   Input: Process tree PT , leaf nodes frequency dictionary Dict_tree
   Output: Dictionary frequencies of all process tree nodes
1  def computeNodeFrequency(PT, Dict_tree):
2      if PT is leafNode() then
3          return Dict_tree[PT];
4      else if T(PT) = XOR then
           // Compute the frequency of XOR node as sum of its children nodes'
              frequencies
5          for child in PT.children do
6              Dict_tree[PT] = Dict_tree[PT] + computeNodeFrequency(child, Dict_tree);
7          end
8          return Dict_tree[PT];
9      else if T(PT) = SEQ or T(PT) = AND then
           // Compute the frequency of SEQ and AND nodes as the frequency of any
              of its children
10         for child in PT.children do
11             computeNodeFrequency(child, Dict_tree);
12             Dict_tree[PT] = Dict_tree[first(PT.children)];
13         end
14         return Dict_tree[PT] ;
15     else if T(PT) = LOOP then
           // Compute the frequency of LOOP node as subtraction of its left and
              right children
16         Dict_tree[PT] = Dict_tree[left(PT.children)] - Dict_tree[right(PT.children)];
17         return Dict_tree[PT];
18 end
```

Applying the above algorithm on the process tree in Figure 5.5 from the event log BPIC_2013_Incident log will result in the decoration of the whole process tree with the frequencies of the nodes of the tree as shown in the example of Figure 5.6 below.

Figure 5.6: Assigned Nodes Frequencies Process tree of BPIC_2013_Incident Log



After inferring the frequencies of the process tree nodes, the next step is to traverse the process tree in a manner respecting the frequencies on the nodes to generate the simulated execution traces from the tree.

### Frequency Constrained Traversal of the Process Tree

After having assigned all the nodes of the process tree with their respective firing frequencies, the next step is to traverse the process tree according to those frequencies. This will allow us to generate simulated traces that are similar in their frequencies of transitions to the inputted Petri net. The traversal of the process tree is done in a top-bottom approach on the tree while decrementing the counts of the nodes traversed until the frequencies are fully satisfied in the process tree. The traversals of the tree must satisfy the counts on the nodes of the tree.

We define the execution order of the nodes in a process tree by our simulation approach as follows:

**Definition 5.1.2** (Execution Order of Nodes in Process Tree by our Approach)**.** Let P be a process tree, T(N) be the type of the tree operator at the root node which can be one of SEQ, XOR, AND, LOOP. Let N be node of a process tree which can be equal to P or any subtree of P that is is a non-leaf node. The execution order of the children of process tree P by our approach is as follows:

1. If T(N) = SEQ : execute leftmost child first followed by second leftmost and so on.
2. If T(N) = AND : execute all the children of the AND node in a fixed order from left to right.
3. If T(N) = XOR: execute the first child of the XOR node with remaining frequency $> 0$
4. If T(N) = LOOP: execute leftmost child, repetition is possible by executing rightmost child additionally then executing the leftmost child again. The overall number of repetitions is equal to the frequency of the rightmost child of the XOR LOOP node.

In figures 5.7 and 5.8 below, we show the frequency-annotated Petri net of the Handling Request For Compensation Log[7] found in section 2.1.2 and its corresponding frequency-annotated process tree as decorated by our approach in the previous section.

Figure 5.7: Frequency Annotated Petri Net Handling Request For Compensation Log

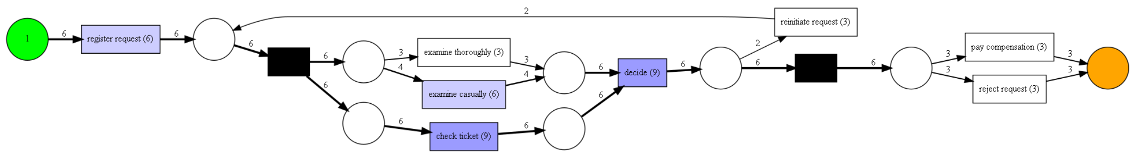
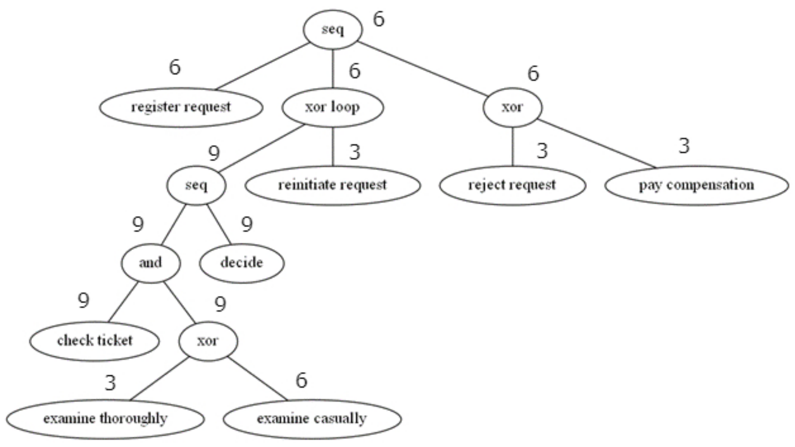


Figure 5.8: Frequency Annotated Process Tree of Handling Request For Compensation Log



We give one example run of the tree traversal execution order in our approach according to definition 5.1.2 for the process tree in figure 5.8. The execution of the process tree should start from the root node and is as follows:

1. root SEQ node is executed

2. children of SEQ node are executed: register request, XOR LOOP, and xor in order from left to right.

3. When XOR LOOP is executed, children of the loop are executed : SEQ, reinitiate request. For each execution of the right child of the loop (reinitiate request), the leftmost child (SEQ) of the XOR LOOP is executed again. That is, leftmost child(SEQ) is executed followed by the execution of AND node, decide node. Next, children of AND node (check ticket and XOR) are executed in left-to-right order. Next, examine thoroughly is executed given its remaining frequency $> 0$; otherwise, examine casually is executed.

4. XOR is executed, and then its child reject request is executed given its remaining frequency $> 0$; otherwise, pay compensation is executed.

The traversal of the process tree is done multiple times until the frequencies of the process tree are satisfied. While nodes are executed, the count of the executed nodes is decremented until the count reaches zero satisfying the frequencies observed - at which the execution is stopped. Therefore, we obtain from the process tree executions that are equivalent in their activity frequencies to the original event log which the Petri net was mined from.

By applying the execution orders in definition 5.1.2, in addition to the frequency constraints on the nodes, we implement the constrained frequency traversal algorithm of the process tree as follows:

**Algorithm 3:** Constrained Frequency Traversal Algorithm

```
Input: Process tree PT , tree nodes frequency dictionary Dict_tree
Output: Execution Traces Dictionary
1  def treeConstrainedTraversalAlgorithm(PT, Dict_tree):
2      Traces ← Initialize
3      while length(Dict_tree[PT]) > 0 do
           // While the frequency of the root node is not zero
4          seq = getExecutionSequence(PT, Dict_tree) // Traverse tree to get one
               trace
5          Traces.add(seq);
6          Dict_tree[PT] = Dict_tree[PT] -1 // Decrement the frequency
7      end
8
```

The function that gets an execution sequence; i,e a sequence of activities, is defined as follows:

**Algorithm 4:** Get Execution Sequence Function

```
Input: Process tree PT , tree nodes frequency dictionary Dict_tree                end
Output: Single Process Tree Execution Trace
1  def getExecutionSequence(PT, Dict_tree):
       // When the node is a leaf node, return the activity node
2      if PT is leafNode() then
3          return [PT];
       // Switch-case by Type of node, get the corresponding sequence of its
           children
4      else if T(PT) = XOR then
5          return getXorSequence(PT,Dict_tree);
6      else if T(PT) = SEQ then
7          return getSeqSequence(PT,Dict_tree);
8      else if T(PT) = AND then
9          return getAndSequence(PT,Dict_tree);
10     else if T(PT) = LOOP then
11         return getLoopSequence(PT,Dict_tree);
12 end
```

XOR Node Execution Sequence as defined in the execution orders in definition 5.1.2.

**Algorithm 5:** Get Execution Sequence XOR

```
Input: Process tree PT , tree nodes frequency dictionary Dict_tree
Output: Execution Sequence of the Process Tree
1  def getXorSequence(PT, Dict_tree):
2      eligible_children ← List
       // Filter Eligible children node by non-zero frequencies
3      for child in PT.children do
4          if Dict_tree[child]>0 then
5              eligible_children.add(child);
6      end
       // Traverse first child node with non-zero frequency to get its
           execution sequence
7      chosen_child = first(eligible_children);
8      Dict_tree[chosen_child] = Dict_tree[chosen_child] - 1;
9      ex_seq = getExecutionSequence(chosen_child, Dict_tree);
10     return ex_seq;
11 end
```

SEQ Node Execution Sequence as defined in the execution orders in definition 5.1.2

**Algorithm 6:** Get Execution Sequence SEQ

```
Input: Process tree PT , tree nodes frequency dictionary Dict_tree
Output: Execution Sequence of the Process Tree
def getSeqSequence(PT, Dict_tree):
    exec_seq ← List
    // Traverse all the children of the node from left to right and get
       their execution sequence
    for child in PT.children do
        if Dict_tree[child] > 0 then
            Dict_tree[child] = Dict_tree[child] - 1;
            exec_seq = exec_seq + getExecutionSequence(child, Dict_tree);
    end
    return ex_seq;
end
```

AND Node Execution Sequence as defined in the execution orders in definition 5.1.2

**Algorithm 7:** Get Execution Sequence AND

```
Input: Process tree PT , tree nodes frequency dictionary Dict_tree
Output: Execution Sequence of the Process Tree
def getAndSequence(PT, Dict_tree):
    eligible_children ← List
    exec_seq ← List
    for child in PT.children do
        if Dict_tree[child] > 0 then
            eligible_children.add(child);
    end
    // Traverse all the children of the node in a fixed left-to-right order
       and get their execution sequences
    for child in eligible_children do
        exec_seq = exec_seq + getExecutionSequence(child, Dict_tree);
        Dict_tree[child] = Dict_tree[child] - 1;
    end
    return ex_seq;
end
```

LOOP Node Execution Sequence as defined in the execution orders in definition 5.1.2

**Algorithm 8:** Get Execution Sequence LOOP

```
Input: Process tree PT , tree nodes frequency dictionary Dict_tree
Output: Execution Sequence of the Process Tree
def getLoopSequence(PT, Dict_tree):
    exec_seq ← List
    repeat ← True
    while repeat do
        if Dict_tree[left(PT.children)] > 0 then
            // Traverse left child node of the Loop first
            repeat ← False
            Dict_tree[left(PT.children)] = Dict_tree[left(PT.children] - 1;
            exec_seq = exec_seq + getExecutionSequence(left(PT.children), Dict_tree) ;
        if Dict_tree[right(PT.children)] > 0 and Dict_tree[left(PT.children)] > 0 then
            // Calculate probability of looping
            loop_probability = Dict_tree[PT] \Dict_tree[left(PT.children)];
            r = random(0,1);
            if Dict_tree[PT] == 1 then
                last_trace ← True ;
            if r ≥ loop_probability or last_trace then
                // repeat the loop according to the probability by traversing
                   the right child node of the Loop
                repeat ← True ;
                Dict_tree[right(PT.children)] = Dict_tree[right(PT.children] - 1;
                exec_seq = exec_seq + getExecutionSequence(right(PT.children), Dict_tree)
                  ;
    end
    return ex_seq;
end
```

**Multiple variant executions and Estimation Approach**

The traversal of a process tree can generate many variant simulations of the same process tree. This is due to the semantics of process trees and their operators as defined in definition 2.1.3. For instance, a parallel operator has the option to generate the execution sequences in multiple orders of the children nodes.A XOR operator has the option to choose any child of the node. A loop operator also can result in different numbers of repetitions in the same trace. Therefore, any combination of a loop, parallel, or XOR operators can result in a hugely variant set of execution traces. In figure 5.8 of the process tree of the running example, multiple scenarios are possible for the AND operator.

For instance, all the following execution sequence orders are possible for the nesting of XOR and AND operator:

1. *<check ticket, examine thoroughly>*
2. *<check ticket, examine casually>*
3. *<examine casually, check ticket>*
4. *<examine thoroughly, check ticket>*

As well as for the loop operator, the following execution sequences are possible by only re-executing the loop with the re-execution of its leftmost child (re-initiate request):

1. *<reinititate request, check ticket, examine thoroughly, decide>*

2. *<reinititate request, check ticket, examine thoroughly, decide, reinitiate request, check ticket, examine thoroughly, decide>*

3. *<reinititate request, check ticket, examine thoroughly, decide, reinitiate request, check ticket, examine thoroughly, decide, reinitiate request, check ticket, examine thoroughly, decide>*

* Further permutations are possible if we consider the permutations of the nested AND and XOR operator.

Therefore, a process tree can have multiple variant executions that are satisfying the frequencies of the nodes of the tree. This was also outlined in chapter4 where we outlined the same challenges in a Petri net.

Our implemented approach, however, does not consider all the possible permutations of the nodes in a process tree. Our approach aims to approximate an execution of the process tree with fixed orders for the AND and XOR operators as defined by the execution orders in definition 5.1.2. Therefore, our approach considers less permutations of the activities in a process tree to try to approximate a best-case scenario of the run of the process tree. A brute force approach as mentioned in chapter 4 would be very computationally expensive as it will have to consider all the permutations of the process tree and take their average.

**Example traversal of Process Tree in Figure 5.8:**

One possible set of traces of the process tree in Figure 5.8 generated using our constrained frequency traversal algorithm and satisfying the frequency conditions is as follows:

1. *<register request, examine thoroughly, check ticket, decide, reject request>*

2. *<register request, examine thoroughly, check ticket, decide, reject request>*

3. *register request, check ticket, examine thoroughly, decide, reinitiate request, check ticket, examine casually, decide, reinitiate request, check ticket, examine casually, decide, reject request>*

4. *<register request, examine casually, check ticket, decide,reinitiate request, examine casually, check ticket, decide, pay compensation>*

5. *<register request, examine casually, check ticket, decide, pay compensation>*

6. *<register request, check ticket, examine thoroughly, decide, pay compensation>*

In the following execution sequences, the frequencies are satisfied as per the process tree where the frequencies of the nodes, i.e corresponding Petri net transitions, are as follows:

< Register request: 6, Examine casually: 6, Examine thoroughly: 3, Check ticket: 9 , Decide: 9, Reinitiate request : 3, Pay compensation : 3, Reject Request : 3>

As noted previously, multiple variations of the execution sequences are possible for the process tree as the loop can still result in many permutations occur in a different permutations in the trace . Therefore, this is not the only outcome that is possible.

**Simulated Execution Traces to Simulated Event Logs**

After the generation of the execution traces, the next step is to transform the traces into an event log. This is step is done by simply adding metadata to the traces generated such as randomly generated timestamps and trace information. The event log is then exported into XES format[26]

After the generation of the simulated execution traces and the corresponding simulated event log, the next step is to apply quantification methods of the re-identification risk from the event log.

## 5.2 Re-identification Risk Calculation from Simulated Event Logs

After the generation of the execution traces and the corresponding simulated event log, we use existing methods to calculate the disclosure risk (re-identification risk) from the event log. Particularly, we adopt the implementation used in Rafiei et al. [5].

To calculate the disclosure risk, Rafiei et al. [5], assume the attacker has some background knowledge that can be exploited to re-identify the trace owner. The background information can be of three types : set, multiset, and sequence.

Rafiei et al. [5] provide an example of background knowledge as follows:

**Definition 5.2.1** (**Background knowledge**[5]). Let L = $[\langle a,b,c,d\rangle^{10}$, $\langle a,c,b,d\rangle^{20}$, $\langle a,d,b,d\rangle^{5}$, $\langle a,b,d,d\rangle^{15}]$ be an event log. $A_L = a,b,c,d$ is the set of unique activities , and $cand^2_{set}(L) = \{\{a,b\},\{a,c\},\{a,d\},\{b,c\},\{b,d\},\{d,c\}\}$ is the set of candidates of the set background knowledge of size 2 , $proj^L_{set}(A) = [\langle a,b,c,d\rangle^{10}$, $\langle a,c,b,d\rangle^{20}$, $\langle a,d,b,d\rangle^{5}$,$\langle a,b,d,d\rangle^{15}]$. For A = $[b,d^2]$ as a candidate of the multiset background knowledge , $proj^L_{mult}(A) = [\langle a,d,b,d\rangle^{5}$, $\langle a,b,d,d\rangle^{15}]$. Also for $\sigma = \langle b,d,d\rangle$ as a candidate of the sequence background knowledge, $proj^L_{seq}(\sigma)$ $= [\langle a,b,d,d\rangle^{15}]$

The strength of background knowledge from the weakest to the strongest w.r.t. the type is as follows: set, multiset, and sequence, i.e., given the event log L, $proj^L_{seq}(\langle b,d,d\rangle) \subseteq proj^L_{mult}([b,d^2])$ $\subseteq proj^L_{set}(\{b,d\})$.

Therefore, per each background knowledge type: set, multiset, or sequence, the work defines a possible space of candidates of the background knowledge. For each candidate, it is possible to infer the traces that can be filtered from the event log based on the knowledge from the candidate set. For instance, knowing the candidate background knowledge $\sigma = \langle b,d,d\rangle$ by the possible attacker, it is possible to filter the event log L traces and project the possible traces corresponding to that background knowledge. Since we know b,d,d activities occurred in that order, then the only corresponding traces from the event log L are $proj^L_{seq}(\sigma) = [\langle a,b,d,d\rangle^{15}]$ since it is the only trace that has b,d,d in that order. Similarly, with the multiset candidate knowledge example $[b,d^2]$ candidate , we know that b occurred once, and d occurred twice; then we can filter the possible traces corresponding to that background knowledge that are $proj^L_{mult}(A) = [\langle a,d,b,d\rangle^{5}$, $\langle a,b,d,d\rangle^{15}]$ since they are the only traces that have b occurred once and d twice.

Based on the background knowledge defined, Rafiei et al. [5] defines two measures to calculate the disclosure risk of an event log: identity(case) disclosure and attribute(trace) disclosure.

**Definition 5.2.2** (**Identity(Case) Disclosure**[5]). We use the uniqueness of traces w.r.t. the background knowledge of size l to measure the corresponding case disclosure risk in an event log. Let L be an event log and type $\in \{set, mult, seq\}$ be the type of background knowledge. The case disclosure based on the background knowledge type of size l is calculated as follows:

$$cd^l_{type}(L) = \sum_{x \in cand^l_{type}(L)} \frac{\frac{1}{|proj^L_{type}(x)|}}{|cand^l_{type}(L)|}$$

$\frac{1}{|proj^L_{type}(x)|}$ measures the possible uniqueness for a candidate of the background knowledge. If given a candidate, the projected information from the trace is a single trace with count 1 (for example , $b,c,d\rangle$, then the candidate background knowledge corresponds to a unique trace in the event log, which is the worst case as the trace owner can be re-identified in that case. The identity case disclosure risk measures for all the possible candidates of a given background knowledge type, the average uniqueness in an event log; therefore, it allows to estimate the average identity(case) disclosure risk based on the uniqueness of the information that can be inferred from the event log given the type of background knowledge and its size.

**Definition 5.2.3** (**Attribute (Trace) Disclosure**[5]). We use the entropy of matching traces w.r.t. background knowledge of size l to measure the corresponding trace disclosure risk in an event log. Let L be an event log and type type $\in \{set, mult, seq\}$ be the type of background knowledge. The trace disclosure based on the background knowledge type of size l is calculated as follows:

$$td^{l}_{type}(L) = 1 - \sum_{x \in cand^{l}_{type}(L)} \frac{\frac{ent(proj^{L}_{type}(x))}{max\_ent(proj^{L}_{type}(x))}}{|cand^{l}_{type}(L)|}$$

$max\_ent(proj^{L}_{type}(x))$ is the maximal entropy for the matching traces based on the type and size of background knowledge.

The trace disclosure risk quantifies the risk of re-identifying a trace in the event log. The measurement of this risk is done based on the entropy (uncertainty) that given a candidate background knowledge we can know the exact trace corresponding to that background information candidate. For instance, given the candidate $\sigma = \langle b, d, d \rangle$, we know that the projected trace of the event log is $proj^{L}_{seq}(\sigma) = [\langle a, b, d, d \rangle^{15}]$. Although this would mean only 1/15 uniqueness given the $\sigma = \langle b, d, d \rangle$ background knowledge; however, we know that the trace corresponding to this background knowledge is only the trace $\langle a, b, d, d \rangle$. Therefore, the entropy , or in other words the uncertainty is 0 in this case since we know for a fact that it is the only trace corresponding to this background knowledge. In this worst case scenario, we know that trace can be disclosed given this information since there is no uncertainty about the trace. The average trace disclosure in the event log is measured based on the entropy of the traces given a certain type and size of background information.

# Chapter 6

# Results & Evaluation

In this chapter, we discuss the results and the evaluation of our approach. Firstly, we explain the experimental setup and our experiments' objective. Secondly, we describe the event logs used in the experiments. Thirdly, we present our evaluation methodology used to validate our results against our hypotheses. Finally, we present the disclosure risks results and discuss the interpretation of the obtained results and validate them against our evaluation criterion.

## 6.1 Experimental Setup

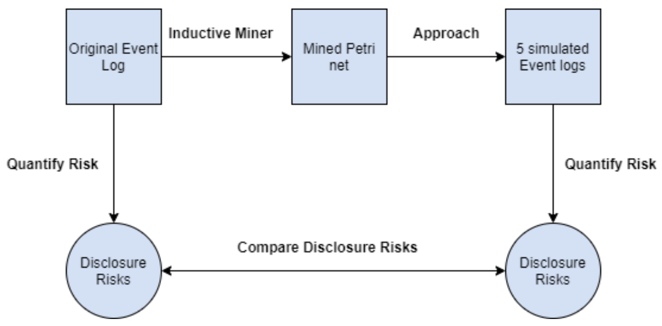
In this section, we demonstrate the experimental setup to obtain the results of our implemented approach. In our experiments, we consider 5 real life event logs which are publicly available at the 4TU Centre for Research Data.[1] The event logs are Sepsis-Cases [27], subset of BPI Challenge 2017 - Offer log [28], subset of Road Traffic Fine Management Process [29], BPI Challenge 2020: Request For Payment and Prepaid Travel Costs logs [30].

For each of the event logs, we generate the frequency-annotated Petri net using inductive miner algorithm ([1], [2], [3]).

From the generated process model, we generate using our approach in section 5, 5 simulated event logs that correspond to the Petri net generated from the original event log. We did not opt for more simulations since there is little variation between the results of the simulated event logs. This is implemented in Python and the implementation code skeleton can be found in *transform_petri_nets_to_event_logs.py* in the GitHub repository of this thesis.[2]. Afterwards, we calculate the identity (case) disclosure and trace disclosure measures as mentioned in our approach in chapter 5 and implemented in Rafiei et al. in [5] using *p-privacy-qt* library published by their work. The quantification risk experimental code can be also found in *quantification_of_risk.py* in our GitHub repository. We report the case disclosure and trace disclosure for both the original event log and the 5 simulated event logs generated from the mined Petri net of the original event log. We show an overview of our experimental setup in figure 6.1

[1]https://data.4tu.nl/
[2]`https://github.com/Karimmaatouk15/quantification_reidentification_risk_process_models/`

Figure 6.1: Overview of Experimental Setup



**Experiment's Objective:** The objective behind the mentioned experimental setup is to compare the identity(case) disclosure and the trace disclosure risks between publishing an original event log and the risk from publishing a Petri net mined from the event log. Therefore, when comparing risk of the logs simulated from the mined Petri net using our approach, we can estimate the risk in the mined Petri net.

## 6.2 Event Logs Description

In this section we describe the event logs that we use in our experiments. In table 6.2, we show statistics of the event logs used in our experiments. #Traces is the total number of traces in the event log. #Variants is the number of unique traces found in the event log. #uniq_activities is the number of unique activities in an event log. Uniqueness describes the percentage of unique traces in the log and is calculated as follows: $uniqueness = \frac{\#Variants \times 100}{\#Traces}$

Table 6.1: Statistics of Event Logs Used in Experiments

| **Event Log** | **#traces** | **#variants** | **#uniq_activities** | **uniqueness** |
|---|---|---|---|---|
| Sepsis-Cases [27] | 1050 | 846 | 16 | 80% |
| BPIC 2017-Offer(Subset) [28] | 2150 | 16 | 8 | 1% |
| Road Traffic Fine(Subset) [29] | 7682 | 231 | 11 | 3% |
| BPIC 2020: Request For Payment [30] | 6886 | 89 | 19 | 1% |
| BPIC 2020: Prepaid Travel Costs [30] | 2099 | 202 | 29 | 9% |

Sepsis-Cases [27] contains events of sepsis cases from a hospital, BPI Challenge 2017 - Offer log [28] pertains to loan application process of a Dutch financial institution, Road Traffic Fine Management Process contains information about managing road traffic fines. [29], BPI Challenge 2020: Request For Payment and Prepaid Travel Costs logs contain information about travel expense claims [30].

These event logs were chosen since they represent real-life event logs and events in these logs usually correspond to individuals. Since we are studying the re-identification risk in this work, considering real-life event logs that pertain to real individuals constitutes a good experimental setup. The event logs are of varied trace uniqueness, from low trace uniqueness BPIC 2017 Offer log with 1% trace uniqueness to highly unique event logs such as the Sepsis-Cases with 80% uniqueness.

The experiments are run on the original event log and 5 simulated logs generated from the Petri net mined from the original event log while varying two parameters: *background knowledge power size* from 1 to 5 and *background knowledge type* : set, multiset, and sequence.

## 6.3 Evaluation

In this section, we set the evaluation criteria of the results obtained in the previous section. Firstly, we look at the objective of this thesis and research questions of this work. Secondly, we establish a method to evaluate the results with respect to our hypothesis. We reiterate the objective and research questions of our thesis as follows:

**Thesis Objective:** The objective is to quantify the re-identification risk of a published Petri net.

**Research Questions:**

1. How to quantify the re-identification risk from the output of a process discovery algorithm (Petri net)?
2. Is it safer to publish a Petri net than to publish an event log?

Prior to the experiments, we introduced the following hypotheses on the identity(case) disclosure and trace disclosure measures:

**Hypothesis(H1):** The identity(case) disclosure risk of a the simulated event logs generated from the Petri net mined from the original event log cannot be higher than the risk of the original log.

Since a Petri net, theoretically, contains less information than an event log, and since a Petri net can correspond to multiple possible event logs as described in the challenges in sections 4.2.1 and 4.2.2 due to their cyclic and concurrent nature, we lay down the hypothesis that the identity (case) disclosure risk resulting from the Petri net mined from an event log cannot be larger than that of the original event log. This hypothesis is possible since the identity(case) disclosure risk increases with the increase in background knowledge size; therefore, the more information available to the attacker, the higher the case disclosure risk, Since, the simulated logs from the Petri net aim to estimate the risk in a Petri net, they also should be evaluated by the same criterion.

Therefore, this of evaluation should be an indicator if our proposed approach and experiments provide an adequate method for estimating the identity(case) disclosure risk in a Petri net.

With regards to the trace disclosure risk,the trace disclosure risk is high even for weak background information as indicated by Rafiei et al. [5].It is not obvious that the trace disclosure risk is higher for higher background information size. Therefore, it does not follow that when an attacker has more background information w.r.t to an event log that the trace disclosure risk increases.Therefore, the same theory and hypothesis that say the identity(case) disclosure risk of a Petri net mined from an event log cannot be larger than the identity(case) disclosure risk of the original event log cannot be applied to the trace disclosure risk. This is because the two risk measures do not follow the same trend w.r.t the increase in the information to the attacker. Therefore, we will be relying on the identity(case) disclosure evaluation method to evaluate whether the estimation/quantification method using the simulated event logs from the Petri net mined from the original event log can be used to estimate the disclosure risks of the Petri net.

**Hypothesis(H2):** The identity(case) disclosure risk and the trace disclosure risk of a the simulated event logs generated from the Petri net mined from the original event log should not be hugely variant between the simulated event logs

Since the simulations' objective is to quantify the disclosure risks, there should not be huge variations between the disclosure risks of the simulated event logs themselves. Otherwise, the quantification method would produce different results for each simulated event log.

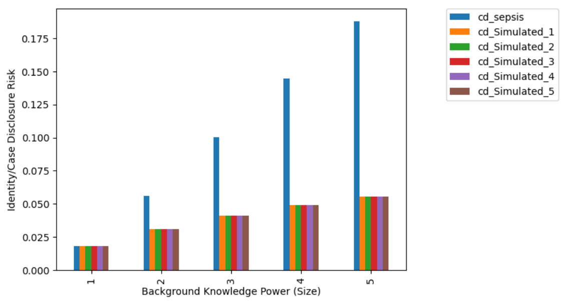
## 6.4 Identity (Case) Disclosure Results

In this section, we report the results of the experiments to quantify the identity(case) disclosure risk in the original event log and the 5 simulated event logs generated by our approach from the Petri net mined from the original event log.

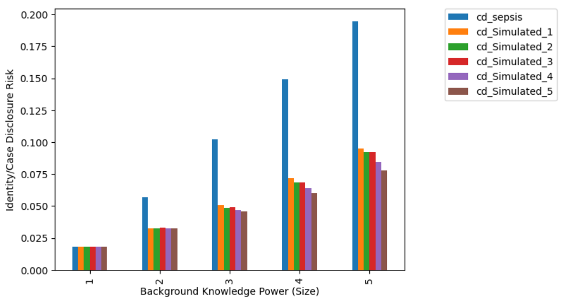
In figure 6.2, we demonstrate the identity (case) disclosure risk on the original Sepsis-cases event log and 5 simulated logs generated by our approach to estimate the risk of publishing a Petri net mined from the original event log. The identity(case) disclosure risk increases with increasing the background knowledge power size. The more background information available to the attacker, the more the risk of re-identifying individuals in the event log. The risk also increases with varying the background knowledge type from set to multiset and sequence respectively which is also explained by more background information available to the attacker about the activities. This increase is noticeable in all event logs that are studied in the experiments that can be seen in table 6.2. The results of the other event logs can be seen in section A of the Appendix.

As can be seen also in figure 6.2, the identity (case) disclosure risk in the 5 simulated event logs generated from the Petri net mined from the original Sepsis-Cases event log is less than or equal to the identity(case) disclosure risk in the original event log with the gap between the original log and the simulated logs increasing with the increase in the background knowledge power size. The risk of the simulated logs from the Petri net is an estimator, in our approach, of the identity(case) disclosure risk of the Petri net mined from the original event log. [3]

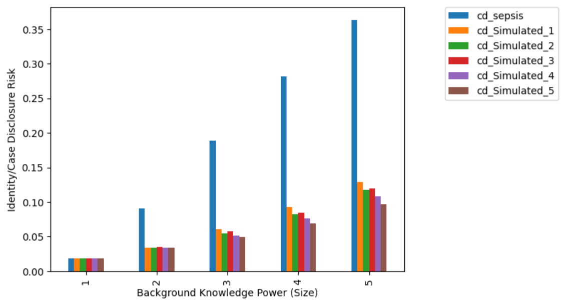
We can also notice that the identity(case) disclosure is not hugely variant between the simulated event logs themselves which is true for all the studied event logs.



(a) Background Knowledge Type: Set



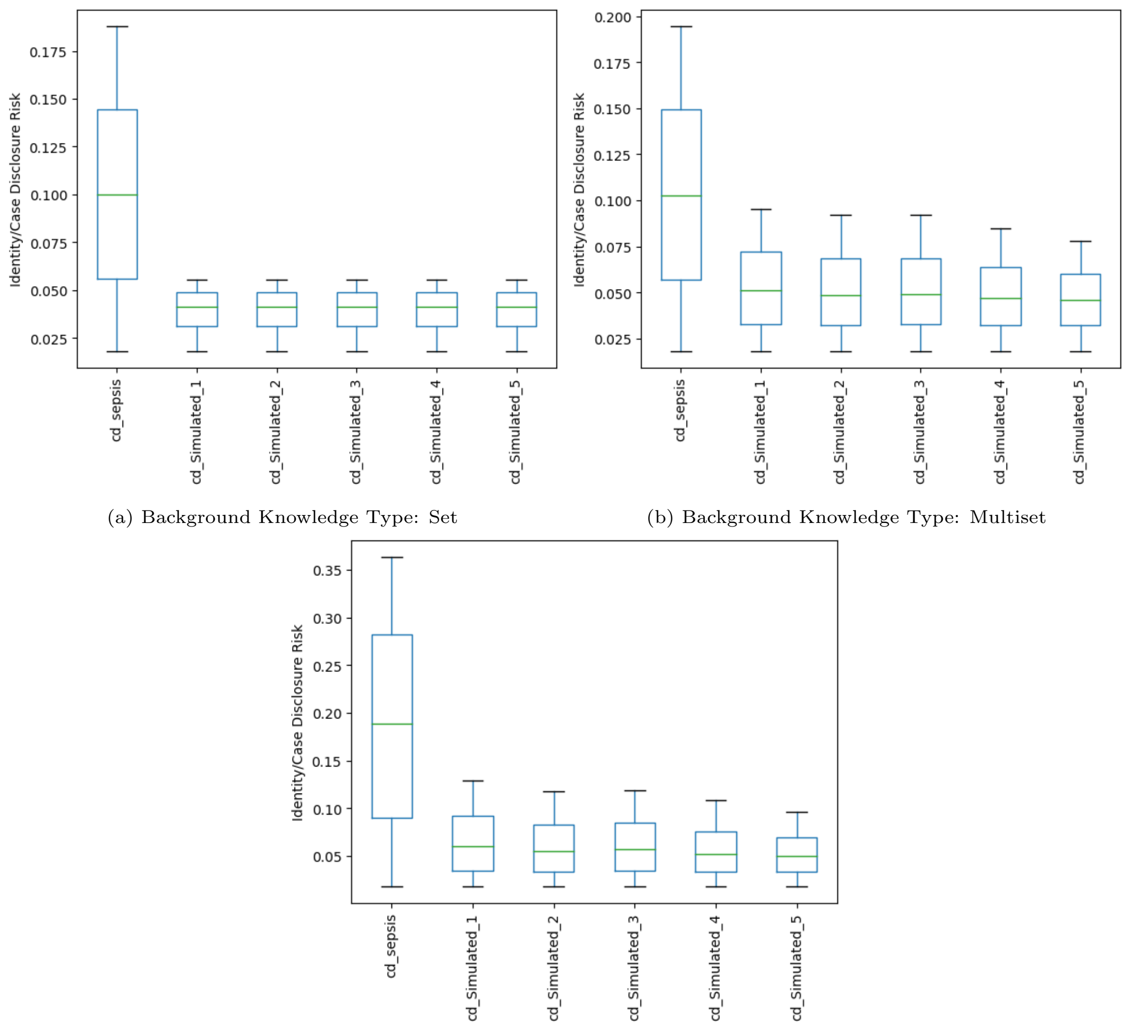
(b) Background Knowledge Type: Multiset



(c) Background Knowledge Type: Sequence

Figure 6.2: Sepsis Case Disclosure in Original Event Log and Simulated Event logs from mined Petri net by Background Knowledge Power Size and Background Knowledge Type

[3]The results of the identity(case) disclosure risk of the other 4 event logs in table6.2 can be found in the Appendix A



(a) Background Knowledge Type: Set

(b) Background Knowledge Type: Multiset

(c) Background Knowledge Type: Sequence

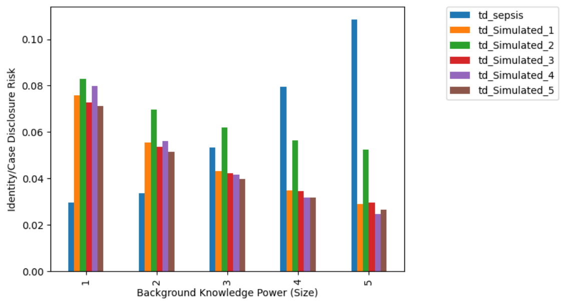
Figure 6.3: Sepsis Case Disclosure in Original Event Log and Simulated Event logs from mined Petri net

## 6.5 Trace Disclosure Results

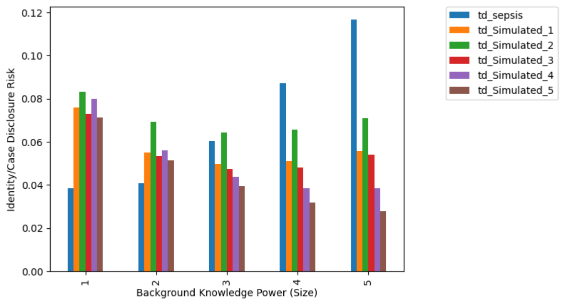
In this section, we report the results of the experiments to quantify the trace disclosure risk in the original event log and the 5 simulated event logs generated by our approach from the Petri net mined from the original event log.

In figure 6.4, we notice that the trace disclosure risk increases for the original sepsis-cases event log but not for the simulated event logs by the increase in the background knowledge power size. However, for other event logs such as the Road Traffic Fine Management event log in figure A.15, we the trend is different, we notice that the trace disclosure risk decreases for the original Road Traffic log and is variant for the simulated event logs by the increase in the background knowledge. This indicates that the trace disclosure, indeed, does not follow the same trend as the identity(case) disclosure risk with the increase in the size of the background knowledge power. The trace disclosure, then, can be high even for weaker background knowledge power size which also aligns with the analysis in Rafiei et al. [5]. The trace disclosure of the 5 simulated event logs is an estimator, in our approach, of the trace disclosure of the Petri net mined from the original event log. [4]

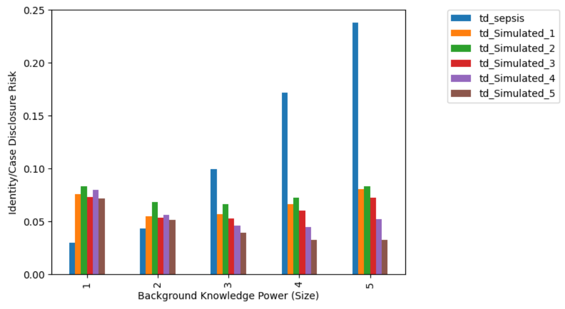
We can also notice that the trace disclosure is not hugely variant between the simulated event logs themselves which is true for all the studied event logs.



(a) Background Knowledge Type: Set



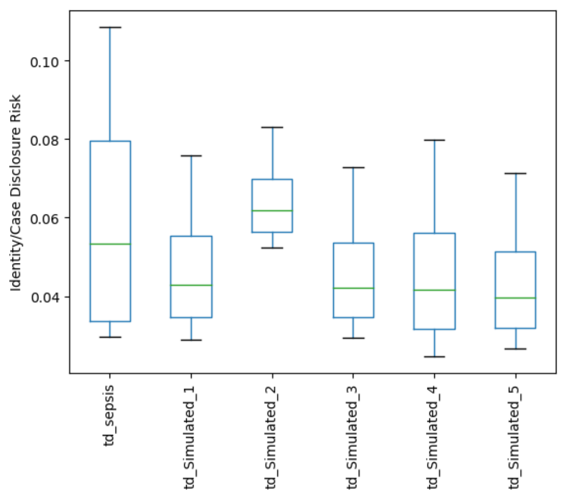
(b) Background Knowledge Type: Multiset



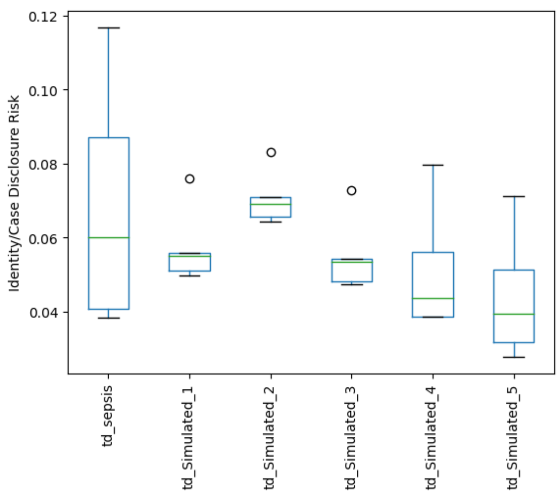
(c) Background Knowledge Type: Sequence

Figure 6.4: Sepsis Trace Disclosure in Original Event Log and Simulated Event logs from mined Petri net by Background Knowledge Power Size

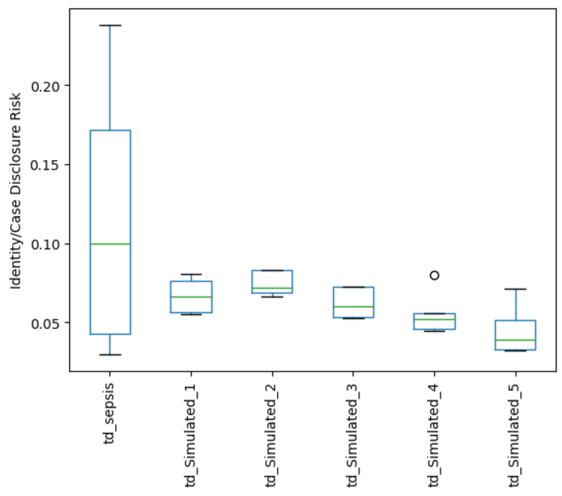
[4]The results of the trace disclosure risk of the other 4 event logs in table 6.2 can be found in the Appendix A



(a) Background Knowledge Type: Set



(b) Background Knowledge Type: Multiset



(c) Background Knowledge Type: Sequence

Figure 6.5: Sepsis Trace Disclosure in Original Event Log and Simulated Event logs from mined Petri net

## 6.6 Discussion

As per the results in the previous sections, the distributions of the identity(case) disclosure risks of the 5 simulated logs of the Petri net mined from the Sepsis-cases log are all below or equal to the risk of the original event log as can be seen in figure 6.3. This result is observed for all event logs, we can notice in the box plots in figures A.2, A.6 , A.10, A.14 that most of the distribution of the simulated logs has less identity disclosure risk than the original logs. Therefore, for all the studied event logs, on average, the identity(case) disclosure risk is less in the simulated logs from Petri net than the original event log which the Petri net was mined from for the logs: Sepis-cases[27], subset of BPIC 2017 event log [28], subset of Road Traffic Fine Management [29],and BPIC2020 event logs: Request for Payment and Prepaid Travel cost [30].

The results in the previous section, therefore, are in line with our hypothesis (H1) in section 6.3 that is:

*The identity(case) disclosure risk of a the simulated event logs generated from the Petri net mined from the original event log cannot be higher than the risk of the original log.*

As we also noted, the identity(case) disclosure risk and trace disclosure risk are not hugely variant between the simulated event logs themselves which is true for all the studied event logs. This goes in line with our hypothesis (H2) in section 6.3.

With this result, we can accept the validity of the method to estimate the disclosure risks for the 5 studied event logs in section as per the evaluation criterion and our hypotheses in section 6.3.

However, the results of the experimentation on the 5 event logs does not guarantee that the hypotheses will be fulfilled for all event logs nor does it validate the correctness of our approach as a method of quantifying the risk measures. Our approach only attempts to estimate the risk and accepts or rejects the results based on the hypothesis provided in our evaluation section.

Finally, we revisit our third research question (RQ3). We already discussed from a theoretical perspective that a Petri net reveals less information than an event log. Thus, it is safe to say that Petri nets are safer to publish generally than the event logs they were mined from. Moreover, the experiments also show that the identity(case) disclosure risk, or re-identification risk, is significantly less, on average, in Petri nets than the original event logs they were mined from. In some cases, however, the re-identification risk can be equal or slightly lower for some background knowledge sizes as shown in the results of our experiments. Therefore, a Petri net can also have an equivalent risk in some cases similar to publishing an event log; however, in practice, generally and on average, they are safer to publish than the event logs they were mined from.

# Chapter 7

# Conclusions

In this chapter, we present the main conclusions of our research. In section 7.1, we outline a summary of the main contributions of this research. In section 7.2, we discuss the limitations of the proposed approach and possible future work needed in this research area.

## 7.1 Summary of Contributions

In this thesis, we discussed possible privacy attacks that an adversary can mount using a published process model(Petri net) .We also proposed a method to quantify the re-identification risk in published Petri nets that are mined from event logs. Additionally, we explored whether it is safer to publish a Petri net than to publish the event log it was mined from. Therefore, our research addressed three main research questions:

**Research Question (RQ1):** What are the types of risks associated with publishing a process model (Petri net) ?
In chapter 3, we outlined three main privacy attacks : re-identification attack, reconstruction attack, and membership disclosure attack that are applicable also when publishing a process model (Petri net). We concluded that it is also possible for an adversary to mount such attacks using the published Petri net combined with background knowledge the attacker possesses or using other publicly available datasets.

**Research Question (RQ2):** How to quantify the re-identification risk of a process model (Petri net) mined using a process discovery algorithm(Inductive miner) from an event log ?
To answer this question, we proposed an approach in chapter 5 that generates simulated event logs from published Petri nets in order to estimate the re-identification risk of these Petri nets. Our approach continues to quantify the risk from these simulated event logs using existing quantification methods. Our experiments and results in section 6.3 on 5 real-life event logs allowed us to accept the validity of our method to estimate the disclosure risks on these 5 event logs as the results were in line with our hypotheses (H1 and H2) presented in the evaluation section 6.3.

**Research Question (RQ3):** Is it safer to publish a process model(Petri net) than to publish the event log it was mined from ?
We concluded ,in our discussion in section 6.3, that Petri nets are generally safer to publish than the event logs they were mined from. This is because, as mentioned, Petri nets theoretically reveal less information than an event log. This is because they are a generalization of the event log and allow for behavior that is outside the event log sometimes; i.e, traces that are not in the event log as described in the challenges in chapter 4 because of their concurrent and cyclic nature. Therefore, an adversary would not be able to know with certainty the exact traces that belong to the original event log which the Petri net was mined from. Thus, the disclosure risk is theoretically less for

published Petri nets than the event logs they were mined from. This was also further validated by the results obtained in chapter 6.3 where the average identity(case) disclosure risk was higher in the original event log than the identity disclosure risk of all the 5 simulated event logs for all the event logs studied, which is an estimator of the re-identification risk of the Petri net mined from the original event log.

## 7.2 Limitations and Further Research

During the work on this research, we encountered several limitations and considered some assumptions due to the limited time available for implementation. These limitations are listed below with possible further future work.

- **Subset of Petri nets (Sound WorkFlow Nets):** In this work, we only considered a subset of Petri nets known as WF-nets in our approach and implementation. This is because the transformation of a Petri net to its corresponding Process tree is only possible if the Petri net is a sound WF-net in Pm4py process mining library. Sound WF-net do not contain deadlocks and make it more feasible for simulations. This area needs further research to consider more general Petri nets; however, it is already common in the process mining research community to mostly consider WF-nets in implemented algorithms.
- **Petri nets Generated by Inductive Miner:** In this work, we considered only Petri nets that are mined using Inductive miner [1]. This is because our approach makes use of process trees and generates the traces of a Petri net using its corresponding process tree which is an intermediary result of the Inductive miner algorithm. Therefore, in future work, Petri nets that are mined using other process discovery algorithms can also be considered.
- **Frequency-annotated Petri nets:** We already discussed in 4 that quantifying the risk from un-annotated Petri nets could be extremely challenging as there is no information about the traces count to estimate the risk. Therefore, we considered in our approach and research only frequency-annotated Petri nets. Therefore, considering other types of Petri nets such as time-annotated Petri nets or un-annotated could be done in future research.

# Bibliography

[1] S. J. J. Leemans, D. Fahland, and W. 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* (J.-M. Colom and J. Desel, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 311–329, Springer, 2013. 1, 6, 34, 43

[2] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering Block-Structured Process Models from Incomplete Event Logs," in *Application and Theory of Petri Nets and Concurrency* (G. Ciardo and E. Kindler, eds.), Lecture Notes in Computer Science, (Cham), pp. 91–110, Springer International Publishing, 2014. 1, 6, 34

[3] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour," in *Business Process Management Workshops* (N. Lohmann, M. Song, and P. Wohed, eds.), Lecture Notes in Business Information Processing, (Cham), pp. 66–78, Springer International Publishing, 2014. 1, 6, 34

[4] S. Nuñez von Voigt, S. A. Fahrenkrog-Petersen, D. Janssen, A. Koschmider, F. Tschorsch, F. Mannhardt, O. Landsiedel, and M. Weidlich, "Quantifying the Re-identification Risk of Event Logs for Process Mining," in *Advanced Information Systems Engineering* (S. Dustdar, E. Yu, C. Salinesi, D. Rieu, and V. Pant, eds.), Lecture Notes in Computer Science, (Cham), pp. 252–267, Springer International Publishing, 2020. 1, 8, 10, 11

[5] M. Rafiei and W. M. P. van der Aalst, "Towards Quantifying Privacy in Process Mining," in *Process Mining Workshops* (S. Leemans and H. Leopold, eds.), Lecture Notes in Business Information Processing, (Cham), pp. 385–397, Springer International Publishing, 2021. 1, 8, 10, 32, 34, 36, 38

[6] E. P. I. Center, "EPIC - Re-identification." 3

[7] W. van der Aalst, "Process Mining - Data Science in Action." 3, 4, 5, 6, 7, 26, 63, 65

[8] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, pp. 541–580, Apr. 1989. Conference Name: Proceedings of the IEEE. 4

[9] F. Mannhardt, *Multi-perspective process mining Proefschrift.* PhD thesis, 2018. ISBN: 9789038644387 OCLC: 1073819465. 4

[10] "General Data Protection Regulation (GDPR) – Official Legal Text." 8

[11] F. Mannhardt, A. Koschmider, N. Baracaldo, M. Weidlich, and J. Michael, "Privacy-Preserving Process Mining," *Business & Information Systems Engineering*, vol. 61, pp. 595–614, Oct. 2019. 8

[12] C. Dwork, "Differential Privacy: A Survey of Results," in *Theory and Applications of Models of Computation* (M. Agrawal, D. Du, Z. Duan, and A. Li, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 1–19, Springer, 2008. 8

[13] M. Rafiei and W. M. P. van der Aalst, "Practical Aspect of Privacy-Preserving Data Publishing in Process Mining," *arXiv:2009.11542 [cs]*, Sept. 2020. arXiv: 2009.11542. 8

[14] M. Rafiei and W. M. P. van der Aalst, "Privacy-Preserving Data Publishing in Process Mining," *arXiv:2101.02627 [cs]*, vol. 392, pp. 122–138, 2020. arXiv: 2101.02627. 8

[15] M. Rafiei, M. Wagner, and W. M. P. van der Aalst, "TLKC-Privacy Model for Process Mining," in *Research Challenges in Information Science* (F. Dalpiaz, J. Zdravkovic, and P. Loucopoulos, eds.), Lecture Notes in Business Information Processing, (Cham), pp. 398–416, Springer International Publishing, 2020. 8

[16] S. A. Fahrenkrog-Petersen, "PRETSA: Event Log Sanitization for Privacy-aware Process Discovery," p. 8. 8

[17] S. A. Fahrenkrog-Petersen, H. van der Aa, and M. Weidlich, "PRIPEL: Privacy-Preserving Event Log Publishing Including Contextual Information," in *Business Process Management* (D. Fahland, C. Ghidini, J. Becker, and M. Dumas, eds.), Lecture Notes in Computer Science, (Cham), pp. 111–128, Springer International Publishing, 2020. 8

[18] A. Pika, M. T. Wynn, S. Budiono, A. H. M. Ter Hofstede, W. M. P. van der Aalst, and H. A. Reijers, "Privacy-Preserving Process Mining in Healthcare," *International Journal of Environmental Research and Public Health*, vol. 17, p. E1612, Mar. 2020. 8

[19] F. K. Dankar, K. El Emam, A. Neisa, and T. Roffey, "Estimating the re-identification risk of clinical data sets," *BMC medical informatics and decision making*, vol. 12, p. 66, July 2012. 8

[20] L. Rocher, J. Hendrickx, and Y.-A. d. Montjoye, "Estimating the success of re-identifications in incomplete datasets using generative models," *Nature Communications*, 2019. 8

[21] K. E. Emam, F. K. Dankar, R. Vaillancourt, T. Roffey, and M. Lysyk, "Evaluating the Risk of Re-identification of Patients from Hospital Prescription Records," *The Canadian Journal of Hospital Pharmacy*, vol. 62, no. 4, pp. 307–319, 2009. 8

[22] J. Domingo-Ferrer, "Disclosure Risk," in *Encyclopedia of Database Systems* (L. LIU and M. T. ÖZSU, eds.), pp. 848–849, Boston, MA: Springer US, 2009. 8

[23] G. Elkoumy, A. Pankova, and M. Dumas, "Privacy-Preserving Directly-Follows Graphs: Balancing Risk and Utility in Process Mining," *arXiv:2012.01119 [cs]*, Dec. 2020. arXiv: 2012.01119. 8, 9, 10, 11

[24] W. Steeman, "BPI Challenge 2013, incidents," Apr. 2013. Medium: media types: application/x-gzip, text/xml Version Number: 1 Type: dataset. 21, 22, 23, 24, 63

[25] S. J. van Zelst, "Translating Workflow Nets to Process Trees: An Algorithmic Approach," *Algorithms*, vol. 13, p. 279, Nov. 2020. arXiv: 2004.08213. 21

[26] "IEEE Standard for eXtensible Event Stream (XES) for Achieving Interoperability in Event Logs and Event Streams," tech. rep., IEEE. ISBN: 9781504424219. 31

[27] F. Mannhardt, "Sepsis Cases - Event Log," Dec. 2016. Publisher: 4TU.ResearchData Type: dataset. 34, 35, 40

[28] B. van Dongen, "BPI Challenge 2017 - Offer log," Feb. 2017. Publisher: 4TU.ResearchData Type: dataset. 34, 35, 40, 47

[29] M. de Leoni and F. Mannhardt, "Road Traffic Fine Management Process," Feb. 2015. Publisher: 4TU.ResearchData Type: dataset. 34, 35, 40, 47

[30] B. van Dongen, "BPI Challenge 2020," Mar. 2020. Publisher: 4TU.ResearchData. 34, 35, 40, 47

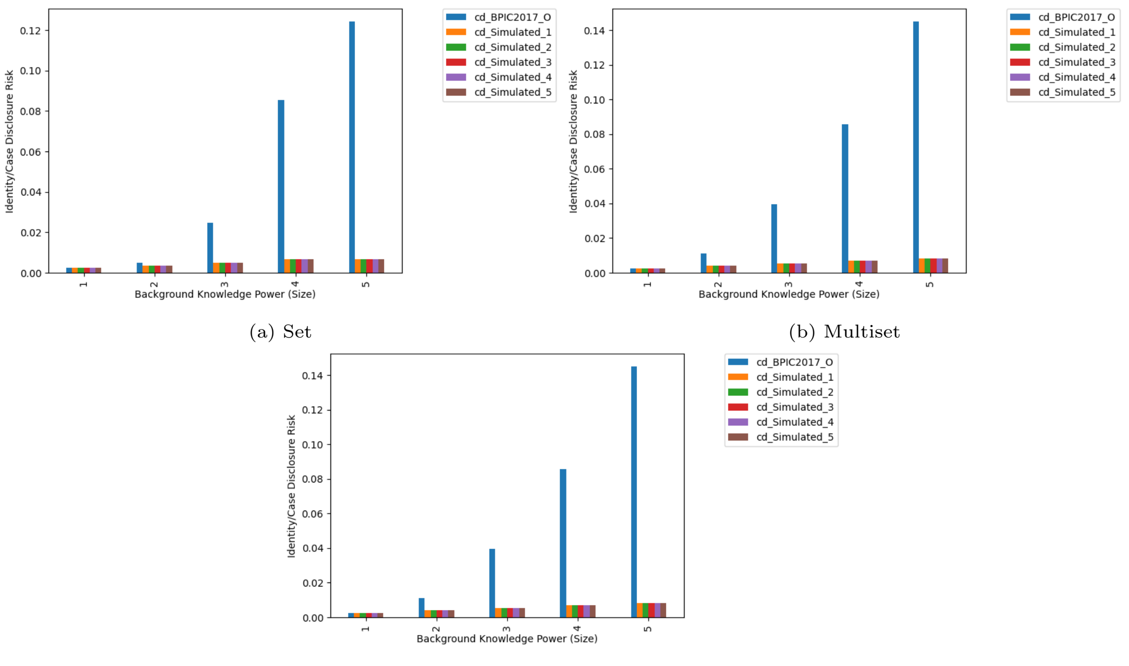
# Appendix A

# Event Logs Case Disclosure and Trace Disclosure

In this appendix, we present the identity(case) disclosure and trace disclosure results of the 4 remaining event logs that are shown in table 6.2. These event logs are: subset of BPIC 2017 event log [28], subset of Road Traffic Fine Management [29],and BPIC2020 event logs: Request for Payment and Prepaid Travel cost [30].

## A.1 BPIC 2017 Event Log

Below we present the case disclosure and trace disclosure results for a subset of the BPIC 2017 event log and 5 simulated event logs generated from the Petri net by our approach to estimate the disclosure risks in the Petri net.

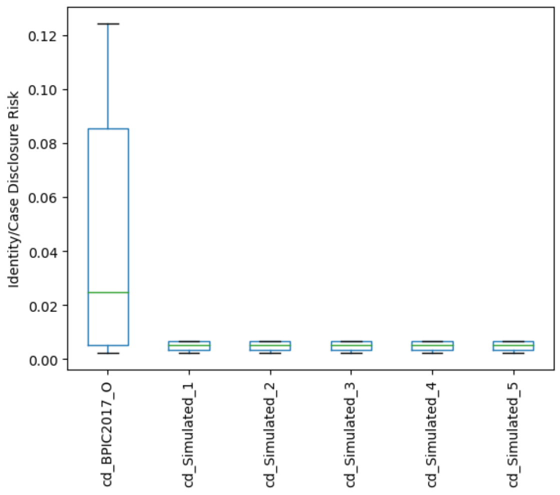


(a) Set

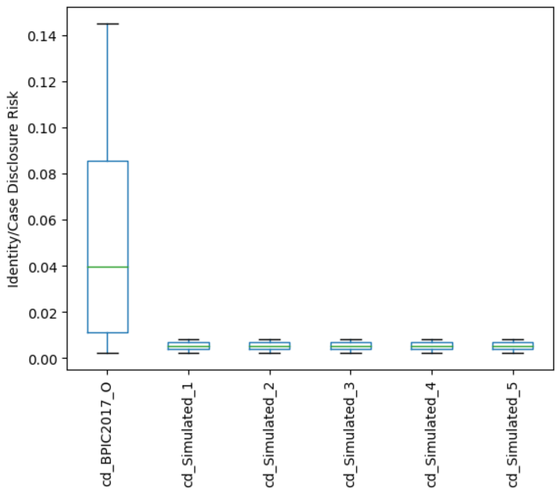
(b) Multiset

(c) Sequence

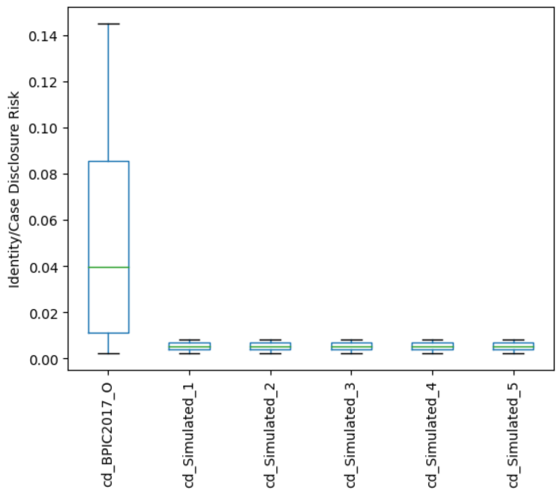
Figure A.1: BPIC 2017 Offers Case Disclosure in Original Event Log and Simulated Event logs from mined Petri net by Background Knowledge Power Size



(a) Set

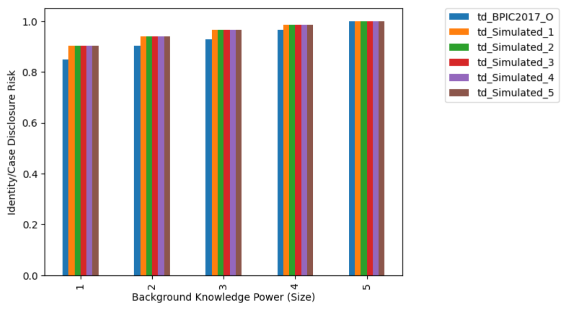


(b) Multiset

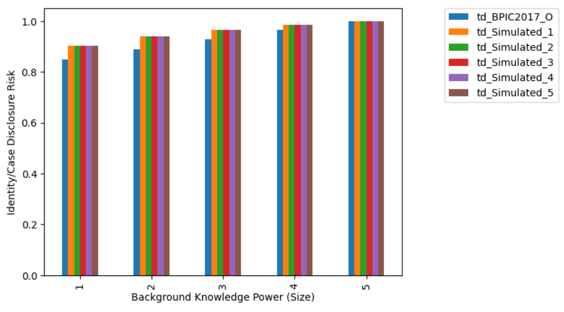


(c) Sequence

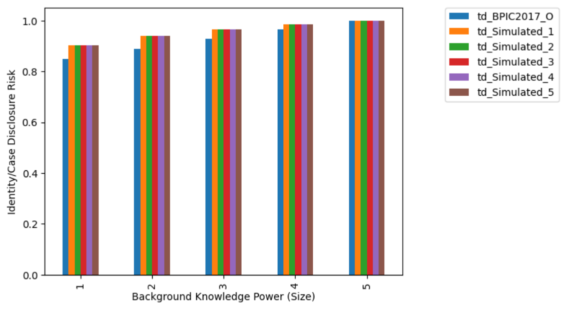
Figure A.2: BPIC 2017 Offer Case Disclosure in Original Event Log and Simulated Event logs from mined Petri net



(a) Set

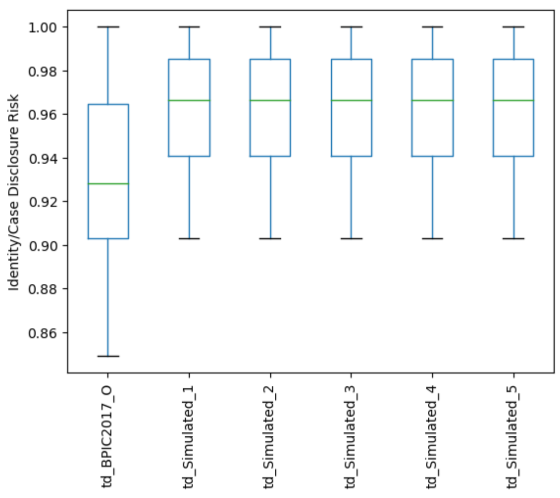


(b) Multiset

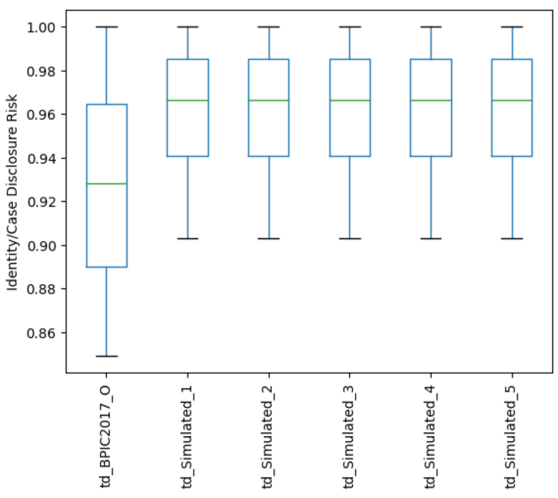


(c) Sequence

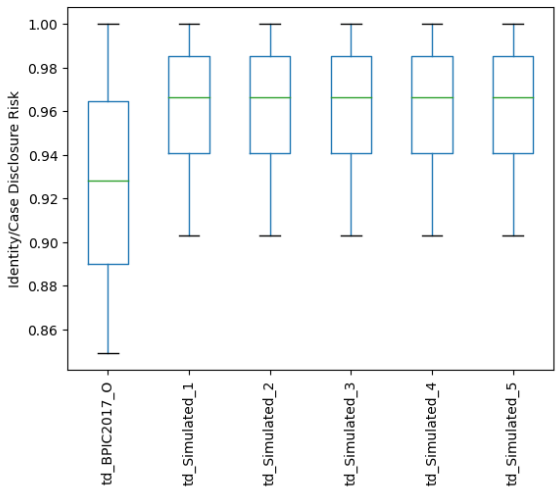
Figure A.3: BPIC 2017 Offers Trace Disclosure in Original Event Log and Simulated Event logs from mined Petri net by Background Knowledge Power Size



(a) Set



(b) Multiset

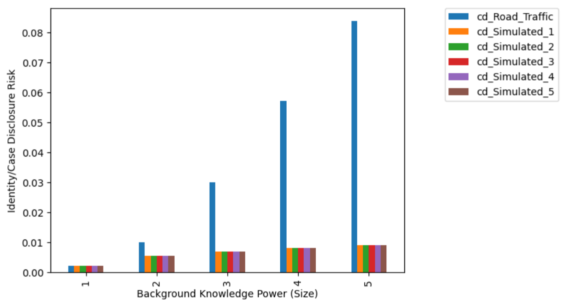


(c) Sequence

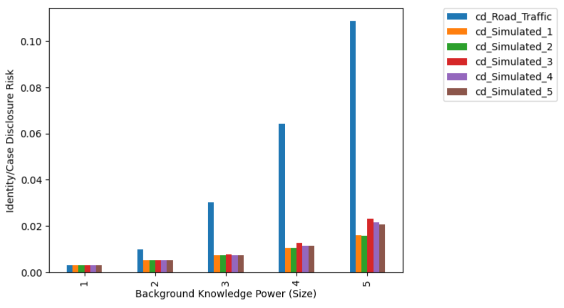
Figure A.4: BPIC 2017 Offer Trace Disclosure in Original Event Log and Simulated Event logs from mined Petri net

## A.2 Road Traffic Fine Management Event Log

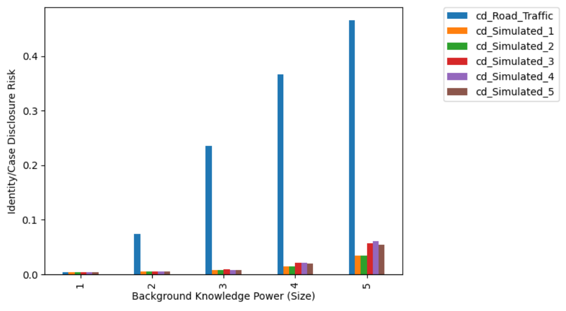
Below we present the case disclosure and trace disclosure results for a subset of the Road Traffic Fine Management event log and 5 simulated event logs generated from the Petri net by our approach to estimate the disclosure risks in the Petri net.



(a) Set

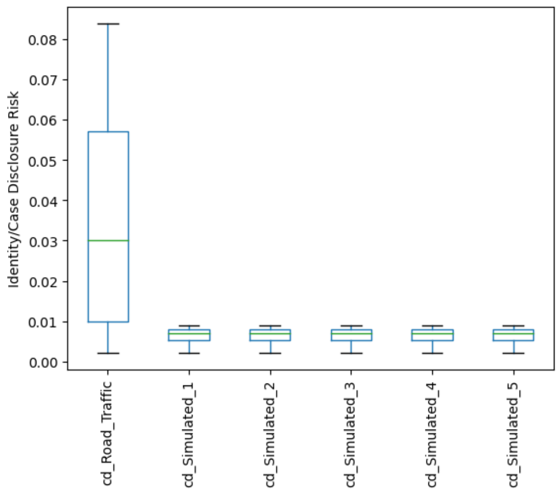


(b) Multiset

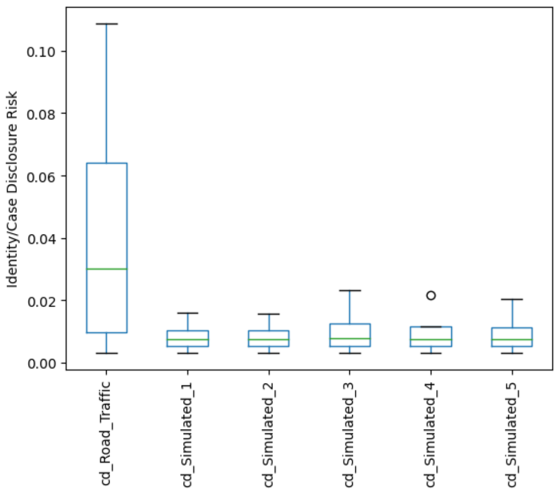


(c) Sequence

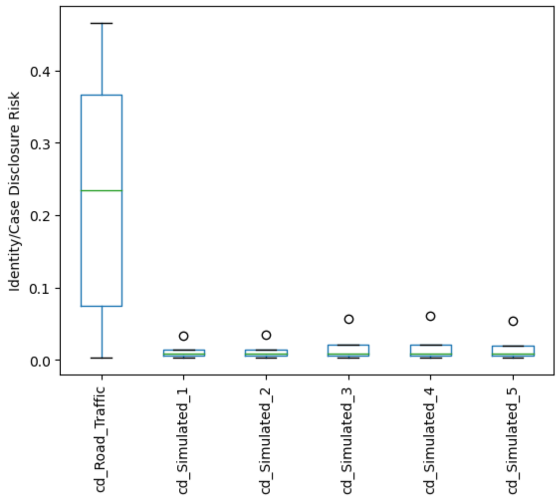
Figure A.5: Road Traffic Case Disclosure in Original Event Log and Simulated Event logs from mined Petri net by Background Knowledge Power Size



(a) Set

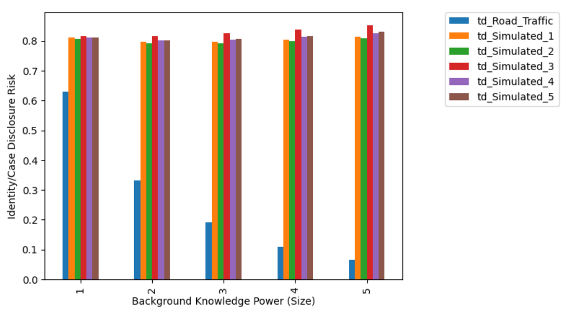


(b) Multiset

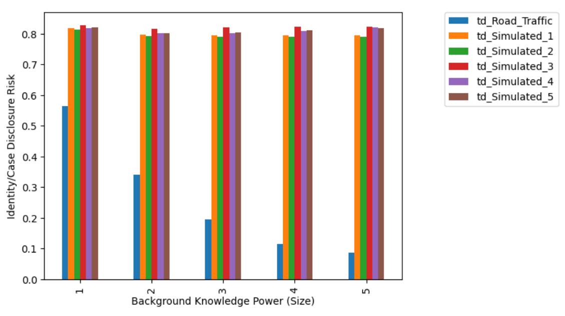


(c) Sequence

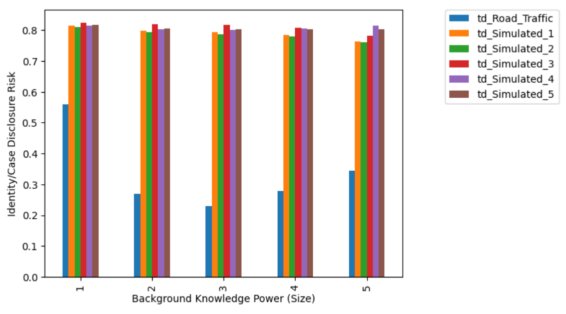
Figure A.6: Road Traffic Case Disclosure in Original Event Log and Simulated Event logs from mined Petri net



(a) Set

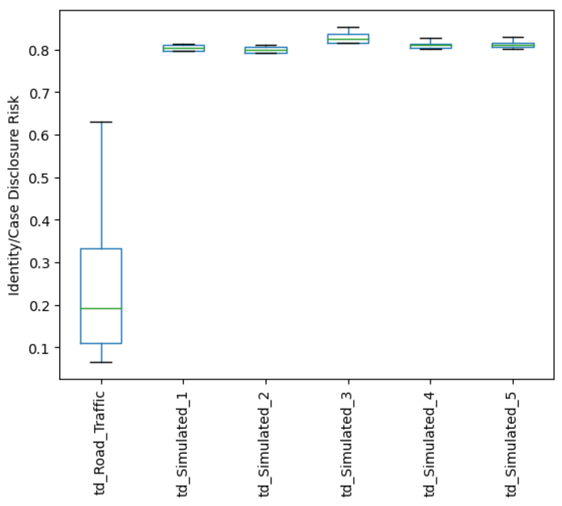


(b) Multiset

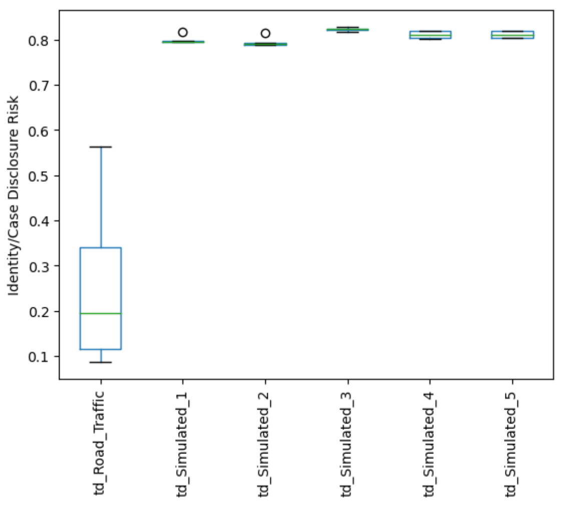


(c) Sequence

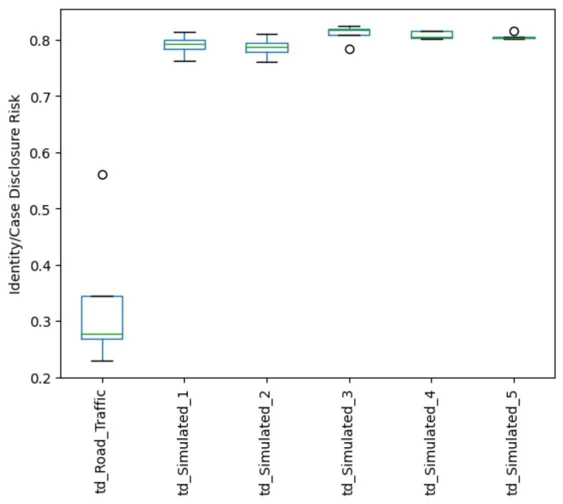
Figure A.7: Road Traffic Trace Disclosure in Original Event Log and Simulated Event logs from mined Petri net by Background Knowledge Power Size



(a) Set



(b) Multiset

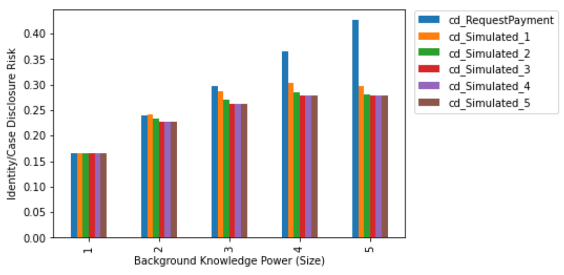


(c) Sequence

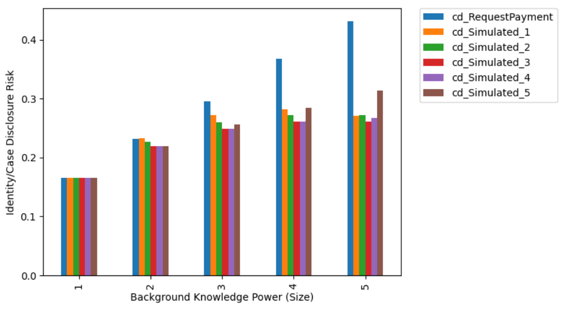
Figure A.8: Road Traffic Trace Disclosure in Original Event Log and Simulated Event logs from mined Petri net

## A.3 Request For Payment Event Log

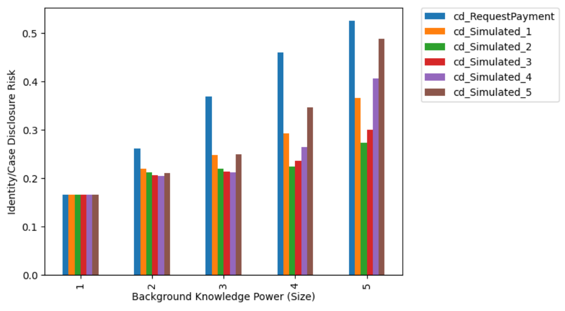
Below we present the case disclosure and trace disclosure results for the Request For Payment event log and 5 simulated event logs generated from the Petri net by our approach to estimate the disclosure risks in the Petri net.



(a) Set

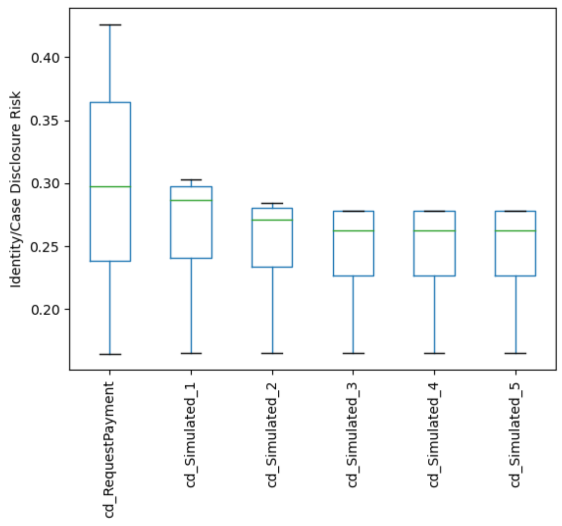


(b) Multiset

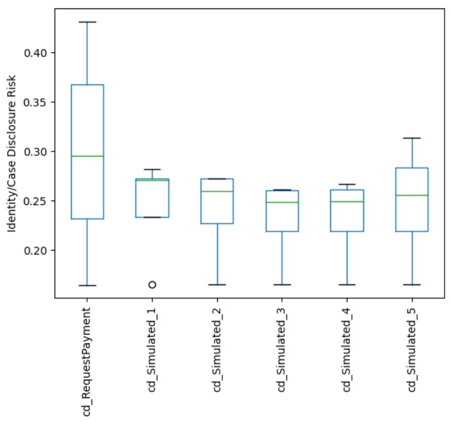


(c) Sequence

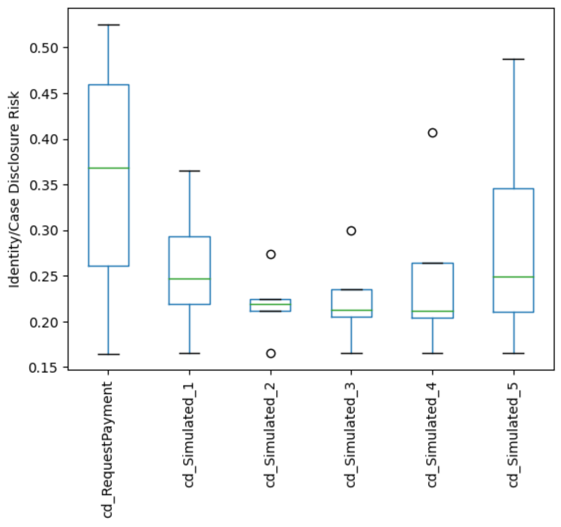
Figure A.9: Request For Payment Case Disclosure in Original Event Log and Simulated Event logs from mined Petri net by Background Knowledge Power Size



(a) Set

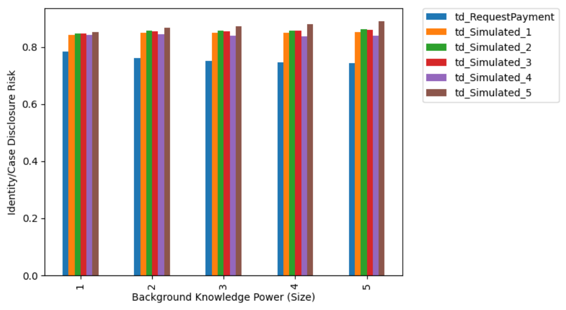


(b) Multiset

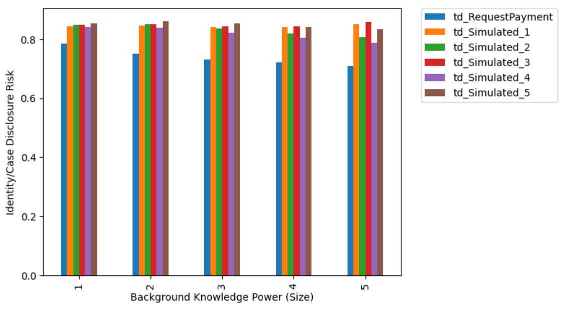


(c) Sequence

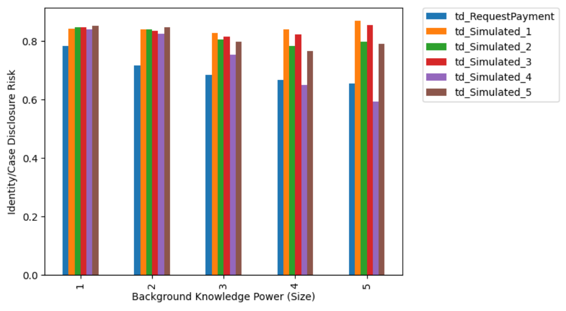
Figure A.10: Request For Payment Case Disclosure in Original Event Log and Simulated Event logs from mined Petri net



(a) Set

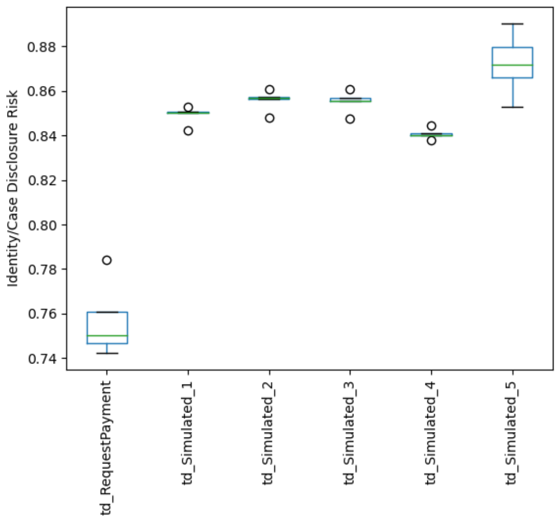


(b) Multiset

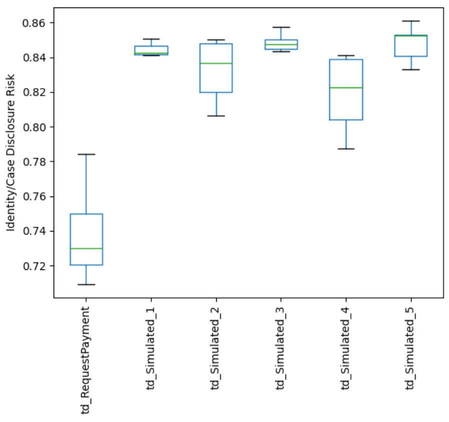


(c) Sequence

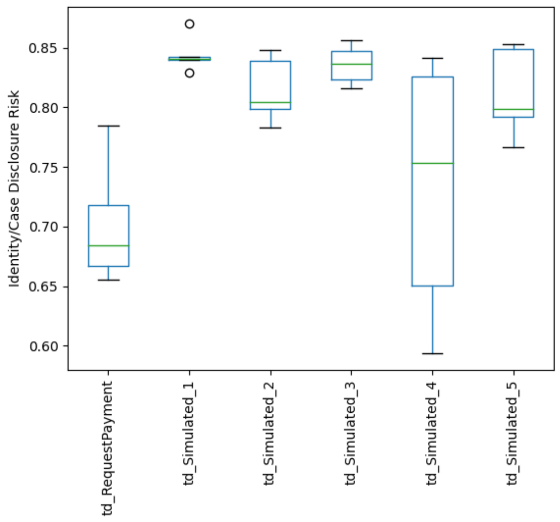
Figure A.11: Request For Payment Trace Disclosure in Original Event Log and Simulated Event logs from mined Petri net by Background Knowledge Power Size



(a) Set



(b) Multiset

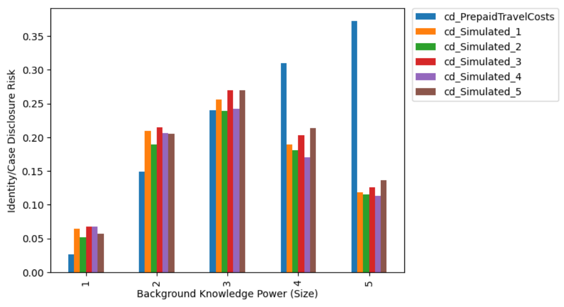


(c) Sequence

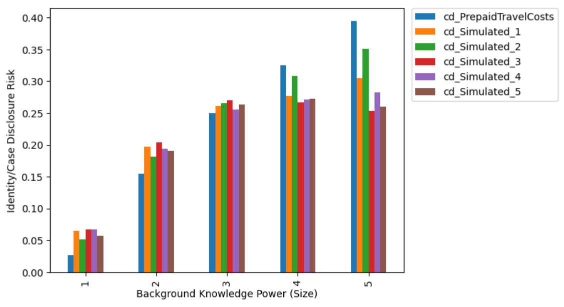
Figure A.12: Request For Payment Trace Disclosure in Original Event Log and Simulated Event logs from mined Petri net

## A.4 Prepaid Travel Cost Event Log

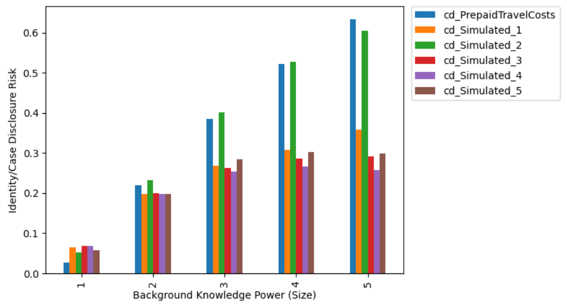
Below we present the case disclosure and trace disclosure results for the Prepaid Travel Cost log and 5 simulated event logs generated from the Petri net by our approach to estimate the disclosure risks in the Petri net.



(a) Set

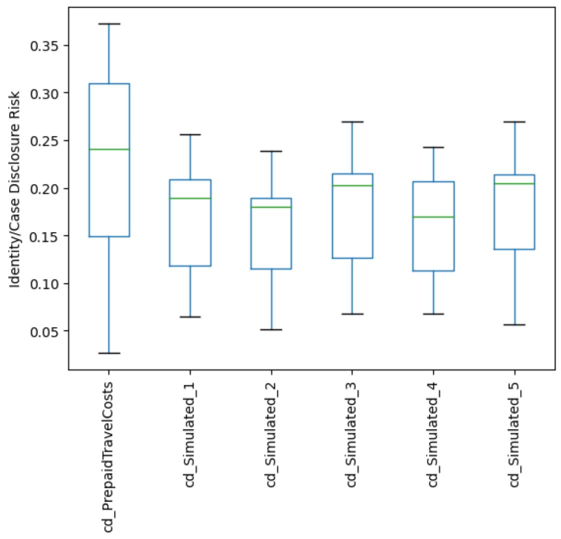


(b) Multiset

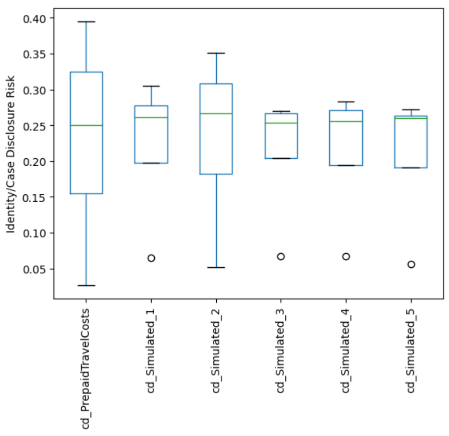


(c) Sequence

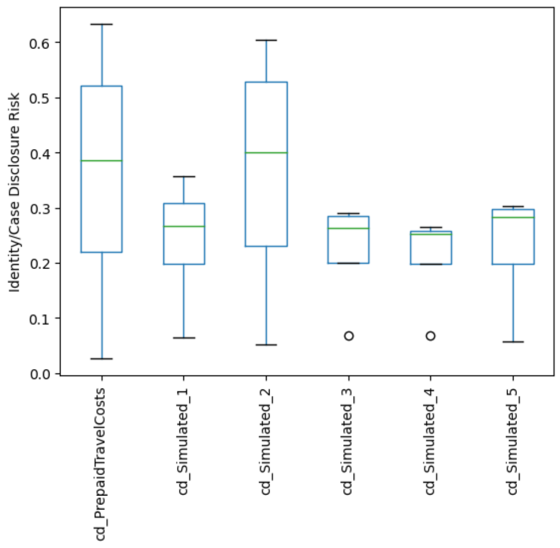
Figure A.13: Prepaid Travel Costs Case Disclosure in Original Event Log and Simulated Event logs from mined Petri net by Background Knowledge Power Size



(a) Set

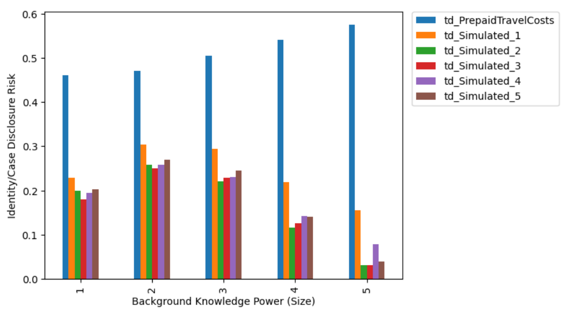


(b) Multiset

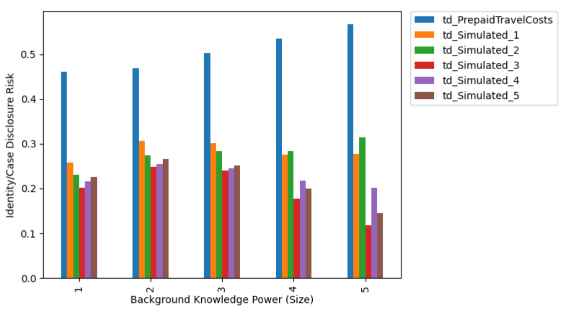


(c) Sequence

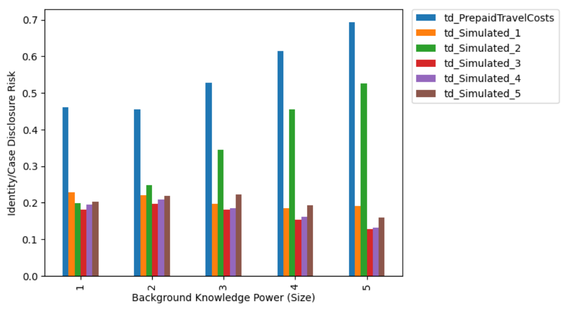
Figure A.14: Prepaid Travel Costs Case Disclosure in Original Event Log and Simulated Event logs from mined Petri net



(a) Set

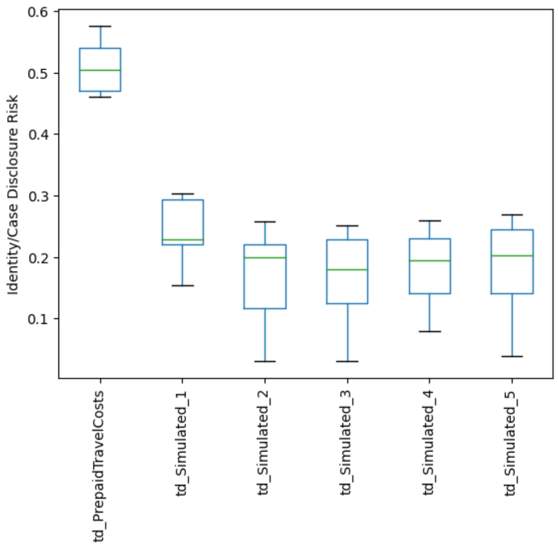


(b) Multiset

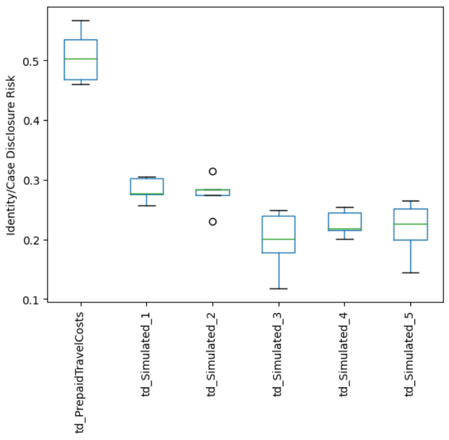


(c) Sequence

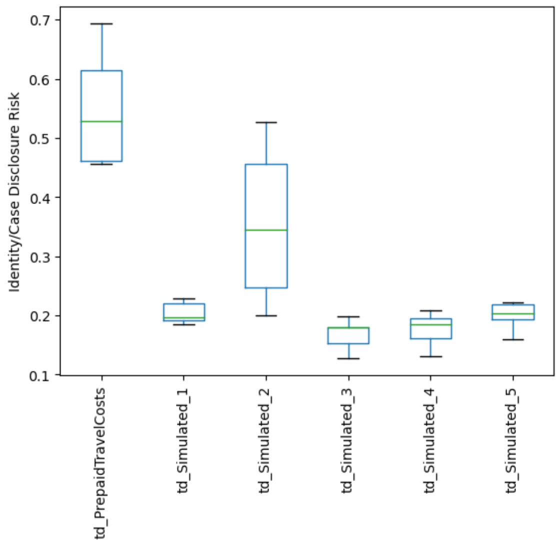
Figure A.15: Prepaid Travel Costs Trace Disclosure in Original Event Log and Simulated Event logs from mined Petri net by Background Knowledge Power Size



(a) Set



(b) Multiset



(c) Sequence

Figure A.16: Prepaid Travel Costs Trace Disclosure in Original Event Log and Simulated Event logs from mined Petri net

# List of Figures

# List of Tables