Universität Ulm | 89069 Ulm | Germany

Faculty of Engineering, Computer Science and Psychology
Databases and Information Systems Department

Applying Process Mining Algorithms in the Context of Data Collection Scenarios

Master's thesis at Universität Ulm

Submitted by:
Marius Breitmayer
marius.breitmayer@uni-ulm.de

Reviewer:
Prof. Dr. Manfred Reichert
Dr. Rüdiger Pryss

Supervisor:
Johannes Schobel

2018

Version from September 26, 2018

Composition: PDF-LaTeX 2_ε

Abstract

Despite the technological progress, paper-based questionnaires are still widely used to collect data in many application domains like education, healthcare or psychology. To facilitate the enormous amount of work involved in collecting, evaluating and analyzing this data, a system enabling process-driven data collection was developed. Based on generic tools, a process-driven approach for creating, processing and analyzing questionnaires was realized, in which a questionnaire is defined in terms of a process model. Due to this characteristic, process mining algorithms may be applied to event logs created during the execution of questionnaires. Moreover, new data that might not have been used in the context of questionnaires before may be collected and analyzed to provide new insights in regard to both the participant and the questionnaire.

This thesis shows that process mining algorithms may be applied successfully to process-oriented questionnaires. Algorithms from the three process mining forms of process discovery, conformance checking and enhancement are applied and used for various analysis. The analysis of certain properties of discovered process models leads to new ways of generating information from questionnaires. Different techniques for conformance checking and their applicability in the context of questionnaires are evaluated. Furthermore, new data that cannot be collected from paper-based questionnaires is used to enhance questionnaires to reveal new and meaningful relationships.

Acknowledgment

I would like to thank everyone who supported me during the creation of this thesis.

I am grateful for the support received from the whole Institute of Databases and Information Systems. It is always a pleasure talking to you.

Prof. Dr. Manfred Reichert, whose lecture on *Business Process Intelligence* sparked my interest in process mining.

I would particularly like to thank Johannes Schobel for his excellent supervision. He always offered me great guidance and challenged me which was invaluable for this thesis.

Thank you to Kevin Andrews and Sebastian Steinau for your support.

A very important thank you also to all those who took the time to proofread this thesis. Your feedback and the time you invested helped to improve this thesis a lot.

Last but not least, I would like to thank my family who gave me the opportunity to study and fully supported me during this time.

Contents

1 Introduction

Even in the age of smartphones and tablets, most questionnaires are still performed via specially tailored "pen & paper" questionnaires. This paper-based approach results in a massive workload for all involved actors. Participants need to be at a certain place to fill in the questionnaire and organizers need to find a location, print the questionnaires, observe the process of filling in the questionnaire, and most importantly transfer the collected data into a format suitable for analysis afterward. As a result, data collection with paper-based questionnaires is not only very time consuming, but also error-prone. Errors may occur when participants answer questions or during the transfer of the collected data. Usually, people copy the paper-based collected data to electronic worksheets by hand. To make the whole data collection process more efficient, *QuestionSys* was developed at the Institute of Databases and Information Systems, Ulm University [1]. *QuestionSys* defines questionnaires in terms of process models and executes them by a process engine [2]. As a result, event logs are generated while a questionnaire is answered, which enables the use of process mining algorithms. Process mining aims to discover, monitor and improve processes by extracting knowledge from event logs. By applying process mining algorithms in the context of data collection scenarios, new insights may be generated in regard to both the structure of a questionnaire as well as the participants.

1.1 Problem statement

When data is collected from questionnaires with *QuestionSys*, the efficiency of the data collection processes is increased [3]. Not only is the efficiency increased, but also new data, e.g. precise temporal information on each question and all answer changes to a question, is collected and documented in event logs. The collection of such precise data from paper-based questionnaires is very difficult and therefore this kind of data is only used in few analyses so far. Oftentimes only rough estimations based on the number of words, questions, decisions or the type of question are conducted. As a result, valuable data which could be used to generate new and meaningful information about the participants of a questionnaire is lost.

The structure of a questionnaire may not always be ideal. Identifying at which point in the questionnaire participants decide to drop out can help to improve its structure and reduce the number of dropouts.

In contrast to questionnaires, there are well-established algorithms to analyze these characteristics in the context of business processes. Process mining offers a variety of different techniques from both process and data science to analyze business processes.

Because *QuestionSys* is based on process technology, these techniques are made available for the analysis of questionnaires. Thus, a questionnaire can be discovered, monitored and improved similar to a business process. Process mining algorithms are often specially tailored towards specific scenarios and the variety of different algorithms is overwhelming.

1.2 Objective

The objective of this thesis is to provide an overview of various process mining algorithms and apply them in the context of questionnaires. To be more precise, process mining algorithms are applied to artificially created event logs. This allows evaluating the applicability of each algorithm in the context of questionnaires. Several process discovery algorithms can discover the control-flow of a questionnaire and indicate new and unknown behavior, e.g. long distance dependencies or loops.

Dropouts can be analyzed using algorithms from conformance checking, and this information may then be used to improve the questionnaire in future versions. Conformance checking algorithms may also be used to quantify the structural change between different questionnaires and questionnaire versions.

Additionally, example scenarios on how to make former unused data usable with the help of process mining algorithms are shown. Although the event logs used in this thesis are not generated by a real questionnaire, the results may still be used as a proof that process mining can be successfully applied to questionnaires.

1.3 Structure of the Thesis

Figure 1.1 shows the structure of the thesis. Chapter 2 provides an overview of scenarios in which process mining has already been applied successfully.

Next, the fundamentals needed to understand the subsequent chapters are introduced in Chapter 3. This includes introductions to event logs, Petri and WorkFlow nets, *QuestionSys*, decision trees and an overview of available process mining tools.

The following chapter, Chapter 4, aims to provide a better understanding of process mining in general. First, a very brief introduction to the three forms of process mining is given and process mining is put into different contexts. Additionally, simplicity, fitness, generalization and precision, are introduced as four competing quality criteria for process mining.

Then Chapter 5 provides an overview of different process discovery algorithms from a control-flow perspective and evaluates their applicability in the context of questionnaires. This chapter includes various algorithms that follow different strategies to discover a process model.

Chapter 6 introduces different conformance checking algorithms. The algorithms focus on footprint comparison, token replay, alignments and an approach based on linear temporal logic.

While Chapter 5 focuses on the control-flow perspective, the focus of Chapter 7 is on algorithms that are able to support different perspectives such as time, resource or data.

Finally, Chapter 8, contains the conclusion of the thesis and an outlook.

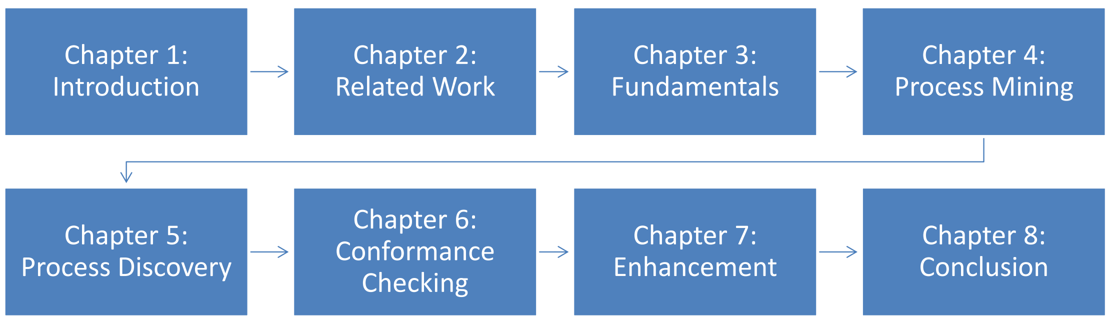


Figure 1.1: Structure of the Thesis

2 Related Work

The objective of this chapter is to give an overview of different scenarios in which process mining was successfully applied.

Mans et al. [4] demonstrate the applicability of process mining in the context of a Dutch hospital. Process mining algorithms are applied to obtain meaningful knowledge about the typical paths followed by particular patient groups within the hospital. Insights on the process was obtained by looking at the control-flow, organizational and performance perspective of the process and initial results are presented.

An approach to assess the efficiency of emergency call centers in France with the help of process mining algorithms is described by Lamine et al. [5]. The effectiveness of the response to an incoming emergency call impacts the quality of each call center. Additionally, the main information used by the French government to distribute their funding is the quality of service. To meet the government's requirements, it is crucial to understand the process. Process mining algorithms are applied to obtain meaningful knowledge about this process. *Disco* was used to discover the control-flow perspective of the process and also identified that the performance mainly depends on certain resources. This knowledge was then used to improve the process.

Van der Aalst et al. [6] apply process mining algorithms in the context of 12 provincial offices of the Dutch National PublicWorks Department. Event logs from an operational WorkFlow Management System supporting the process of invoice handling within the organization are used to illustrate the practical application of process mining. The results of process mining algorithms enabled the management of the department to both formulate and target specific organizational measures and can be used to support these measures. From the process mining point of view, the two most important outcomes are that both the discovery of the main flow and the combination of different perspectives can be used to better understand a process. In addition, it became evident that real-life event logs often contain loops, incompleteness, and noise.

Bala et al. [7] extend the field of process mining towards mining of software development processes. Project managers of software development projects may obtain valuable new insights by extracting process knowledge from the historical data of software artifacts. Mining the time perspective allows them to monitor whether the software is developed according to the predefined plan and also identify which tasks resulted in a delay. Mining the organizational perspective of a software development process can generate valuable insights into the different skill profiles within the project team. Moreover, these profiles can be used to check if any contractual agreements have been violated.

The focus of Andrews et al. [8] is more on the data quality aspect of process mining. The authors apply process mining algorithms on the process of transporting trauma patients to a hospital in Queensland.

A review of 74 different articles on process mining in the healthcare domain is provided in [9]. Rojas et al. analyze each of the case studies based on eleven aspects. These aspects are: process type, data type, frequently posed questions, process mining perspectives, tools, algorithms, methodologies, implementation strategy, analysis strategy, geographical analysis and the medical field. This literature review indicates that process discovery algorithms are mostly used to discover the control flow perspective and that frequently applied algorithms can all adequately deal with noise and incompleteness of event logs. Moreover, the literature review identifies that conformance checking and performance analysis, as well as event logs containing additional data, are some of the emerging trends of process mining.

In [10], Mans et al. show that process mining techniques may be applied successfully to clinical data. The authors use process mining algorithms to get a better understanding of how different groups of patients take different clinical pathways across different hospitals. Comparing the paths across different hospitals allows to discover different treatment practices and also highlight unexpected behavior.

The applicability of process mining to less structured processes is demonstrated by Rozinat et al. in [11]. Process mining algorithms are applied to the test process of wafer steppers in ASML, the leading manufacturer or chip-making equipment. The test process for wafer steppers consists of three phases. After finishing the three phases, the wafer stepper is taken apart, shipped to a customer and reassembled. Afterward and some of the tests are repeated. The nature of wafer steppers being a highly specialized product also changes the structure of the event log used for process mining. Instead of many instances with only a few events, the test process consists of only a few instances with many events. This makes new algorithms necessary that are able to deal with less structured processes.

Process mining can also be used in the context of more flexible processes. Guenther et al. [12] describe two approaches to discover changes within a process. The discovered change processes provide an overview of all changes that happened so far within a process. Process mining can be used to better support truly flexible processes by helping to understand why and when certain changes become necessary.

Under the section *Process Mining Case Studies* [1], the *IEEE CIS Task Force on Process Mining* offers an overview of different case studies on process mining. The case studies cover a variety of industries such as service, manufacturing, healthcare, construction, utility, or chemical. As of today (02.09.2018), the webpage contains 35 case studies. Additionally, an extensive list of process mining applications can be found which contains 117 entries of application scenarios of process mining [2].

The great variety of different scenarios in which process mining algorithms have already been applied successfully proves that process mining is of great relevance when analyzing and improving processes. Although a wide range of different application scenarios is already identified, to the best of my knowledge, there are currently no approaches that use process mining in the context of questionnaires.

[1]`https://www.win.tue.nl/ieeetfpm/doku.php?id=shared:process_mining_case_studies`, last accessed 02.09.2018

[2]`https://www.win.tue.nl/ieeetfpm/lib/exe/fetch.php?media=casestudies:hspi_process_mining_database_correct_v0.2.pdf`, last accessed 02.09.2018

3 Fundamentals

This chapter outlines the relevant background information which is important to understand the rest of this thesis. The first section explains event logs which are the input for process mining algorithms. This section includes prerequisites, metrics that can be derived directly from the event log, established guidelines for logging in the context of process mining and a short overview of logging formats. The second section introduces the concept of nets, especially Petri nets and WorkFlow nets, which are important to understand the output generated by process mining algorithms. Section three briefly introduces the concept of decision trees, which are used in later chapters of this thesis to enhance a process model. The following section provides a short overview of *QuestionSys* which is a system that represents questionnaires as process models, allowing to apply process mining algorithms. In the last section, an overview of different process mining tools is provided.

3.1 Event Logs

The focus of this section is on event logs representing the input side of process mining. The goal of process mining is to discover, monitor and improve real processes while using knowledge which is already available in today's information systems. To do so, event logs are used as an input for a variety of process mining algorithms which are then able to generate and analyze a process model based on the information contained in the event log. This section provides the necessary knowledge to both understand the concept of event logs as well as the fundamentals of logging for later chapters. The first subsection explains the prerequisites, while the second subsection provides process characteristics that may already be derived from the event log itself. Then twelve guidelines are introduced which aim to create event logs that may be used as a good starting point for process mining [13]. In the last subsection, XES is introduced as the de facto standard logging format for process mining.

3.1.1 Prerequisites

Event logs usually serve as the starting point for process mining. A process may be described by a multiset process instances, also called cases or traces. Every process instance represents a concrete execution of the process. Each process instance may then be described by the events or activities which have been executed within this specific instance. To give an example in the context of questionnaires, a process instance may be the questionnaire filled in by a specific user and an activity then corresponds to a specific question answered by that user. In order to reliably use an event log for process mining, it must be possible to order the activities within each process instance. Without ordering the activities in a case it is impossible to discover any causal relationships such as i.e. activity `A` is always followed by activity `B`. As a result, the bare minimum information that has to be stored in the event log are a case identifier and ordered activities which have been executed within this case [14]. If the activities are not or only partially ordered, timestamps captured by the event log may be used to order the activities. Additionally, timestamps are very helpful when analyzing performance-related properties of the process. Different properties may also be derived from the event log itself. A brief overview of these properties is given in Section 3.1.2. If the event log contains more than the minimal information, this additional information can be used to enhance the process model by e.g. calculating the average time needed per activity, or discovering which data influences a specific decision during process execution.

3.1.2 Log metrics

In the context of process mining, it is important to understand that the complexity and size of an event log are determined by different factors. Therefore, multiple event log metrics are needed to adequately characterize an event log used for process mining. These metrics can also be used to compare different event logs of the same process with each other.

A variety of different metrics such as the

1. Number of cases
2. Average/Minimal/Maximal/Standard deviation of each trace
3. Number of distinct activities (also applicable per case)
4. Number of distinct cases
5. Number of events
6. Number of direct successions
7. Number of start/end activities

have been defined in [14] on the pages 364-368, and thus are not further explained in this thesis.

These event log metrics can be used to summarize a considerable amount of data into only a few key figures. Not only do these metrics allow to challenge big logs with many cases and activities, but also allow to get a better understanding of the event log. Please note that the number of distinct activities is often times a lot smaller than other metrics such as the number of cases or the number of events. While some mining algorithms (i.e. the Alpha Algorithm or the Heuristic Miner) are based on a directly-follows graph, iterating through the event log is typically the most time-consuming task during the execution of such an algorithm. As a result, it is important to have a good understanding of the event log prior to starting with the actual mining of the process model.

3.1.3 Guidelines for Logging

This section introduces twelve guidelines which aim to create a good starting point for process mining. They have been established in [13].

Before heading towards the guidelines, some clarification is necessary. No assumptions on the underlying technology used to record an event log is made, and the definitions are rather loose. For this reason, it is possible to apply the guidelines to all event logs related to process mining. Within this subsection events are described by references and attributes and simply refer to "things that happen". A reference has a name and an identifier which both refer to some object [13]. An object could, for example, be a person, a case or a ticket. Attributes consist of a name and a value, e.g. time = "08-05-2018 12:46:21" or name = "Marius".

When using "raw events" to create an event log four steps are necessary.

1. Select all process relevant events
2. Create process instances by correlating events
3. Order events within a process instance
4. Select or compute attributes (resource, data, ...) based on raw data

The following guidelines are not very specific because they aim to improve logging itself. The guidelines should point out problems related to the input of process mining and offer guidance on how to avoid these problems so that the event log can be used to better instrument software [13].

GL1 *Reference and attribute names should have clear semantics.*

This guideline refers to the situation that references and attributes may be interpreted differently by different people involved during creation and analysis of event data. To avoid this, each stakeholder should interpret event data in the same way.

GL2 *There should be a structured and managed collection of reference and attribute names.*

A collection of references and attributes names allows each stakeholder to look up the meaning of them. In an ideal scenario, the names are grouped hierarchically, and an addition to the collection can only be made after there is a general consensus on both its value and meaning. Organization or domain specific extensions can also be realized with extension mechanics provided by formats like XES [15].

GL3 *References should be stable.*

This guideline refers to the problem that unstable references make the analysis unnecessarily complicated. If the identifier in the event log depends on the context or is reused, analyzing the event log gets much more complicated as these dependencies need to be identified and corrected first. For example, the event log identifiers should be the same regardless of the language setting, the region or time of the system.

GL4 *Attribute values should be as precise as possible.*

The more precise a value is, the more reliable it can be used, not only for process mining. If the desired precision cannot be guaranteed, for example, if only a date instead of a timestamp is available, this should be indicated in the event log using a qualifier.

GL5 *Uncertainty with respect to the occurrence of the event or its references or attributes should be captured through appropriate qualifiers.*

Even if this sounds similar to **GL4**, there is a difference between uncertainty and imprecision. Some values may be less reliable due to communication errors, but their precision is still fine.

GL6 *Events should be at least partially ordered.*

Process mining algorithms with a local strategy, like the Alpha Algorithm, use the ordering of the activities within each instance to derive dependencies before generating the process model. As a result, it is important to order the events in the event log, either explicitly via a list, or implicitly through timestamps. It is possible to order events based on observed causalities, for example, if the recorded timestamps are unreliable or imprecise. This is both unnecessary and cumbersome and can be prevented by ordering the event log during collection.

GL7 *If possible, also store transactional information about the event.*

Transactional information of an event may be start, complete, abort, schedule, assign, suspend, resume, withdraw, etc.. This allows for the computation of activity durations. To make such a computation easier, it is recommended to store activity references to relate events belonging to the same activity instance. Leaving out this reference may result in uncertainties about which events belong together, and which start event corresponds to which complete event.

GL8 *Perform regularly automated consistency and correctness checks to ensure the syntactical correctness of the event log.*

Checking for missing references, attributes or names which are not agreed upon (see **GL2**) helps to keep the quality of the event log high over time. Assuring the quality of an event log is a continuous process to maintain a high data quality within the event log.

GL9 *Ensure comparability of event logs over time and different groups of cases or process variants.*

Over time the logging itself should not change. To keep results from process mining comparative, it is essential that the same logging principles are used. Leaving out some events for a group of cases even though they did occur in reality may also lead to differences that do not actually exist. Even worse, this may lead to wrong conclusions based on the event data.

GL10 *Do not aggregate events in the event log used as input for the analysis process.*

The event data contained in the event log should be as *"raw"* as possible, because aggregation can also be done during analysis, but cannot be undone if the aggregation was done earlier. Additionally, aggregating during analysis increases the reproducibility of the generated results and allows to use the same data within different contexts.

GL11 *Do not remove events and ensure provenance.*

Being able to reproduce results is key for process mining. Therefore, deleting objects from the event log may lead to misleading analysis results. Instead of removing the object, marking it as not relevant allows to not take these objects into account while guaranteeing reproducibility. This is called a "soft delete". Instead of deleting a questionnaire instance, it might be aborted, or a concert is canceled, an employee is fired, a student is exmatriculated, a car is scrapped, or an invoice is paid.

GL12 *Ensure privacy without losing meaningful correlations.*

Especially in the context of questionnaires sensitive or private data is collected, and this sensitive information should be removed as soon as possible. No analysis should be conducted while sensitive or private data is still contained in the event log. The challenge is, to find a suitable trade-off between privacy and process model quality. If sensitive information is removed, no correlations should be removed because this would result in bad process models. A suitable trade-off between privacy and analysis may be provided by hashing the sensitive information.

3.1.4 Logging Formats

Being able to fulfill the requirements identified in Section 3.1.3, a data format was needed to store event logs. In 2003 Mining eXtensible Markup Language (MXML) emerged and was adopted for process mining in 2005 [16]. From that point in time, it has been the standard format for storing and exchanging event logs until it was replaced by eXtensible Event Stream (XES) in 2010 [15].

The main purpose of XES is for process mining, however, it is also suitable for data mining, statistical analysis, and text mining. Four guiding principles have been used to design the XES format:

1. **Simplicity** Information should be presented in the simplest possible way. XES logs should be easy to generate and parse while keeping the log readable for humans.
2. **Flexibility** It should be possible to capture event logs from any background.
3. **Extensibility** It must be easy to extend the standard in a transparent way in the future while maintaining both forward and backward compatibility.
4. **Expressivity** To keep the loss of information as minimal as possible, all information elements must be strongly typed.

Figure 3.1 describes the complete meta-model for the XES standard. In comparison to MXML, the XES format is less restrictive and it is easier to extend the format. A XES document contains one log which again contains any number of traces which are described by a sequential list of events that correspond to a case. As described earlier, this is the exact representation of a process. A process log consists of multiple traces, earlier introduced as instances, which again consist of multiple events, also known as activities. Furthermore, does the XES format support the representation of various attributes which can be used to document additional data which is not part of the previously explained minimal information of an event log. For a more detailed description of the XES format, please be revered to [15].

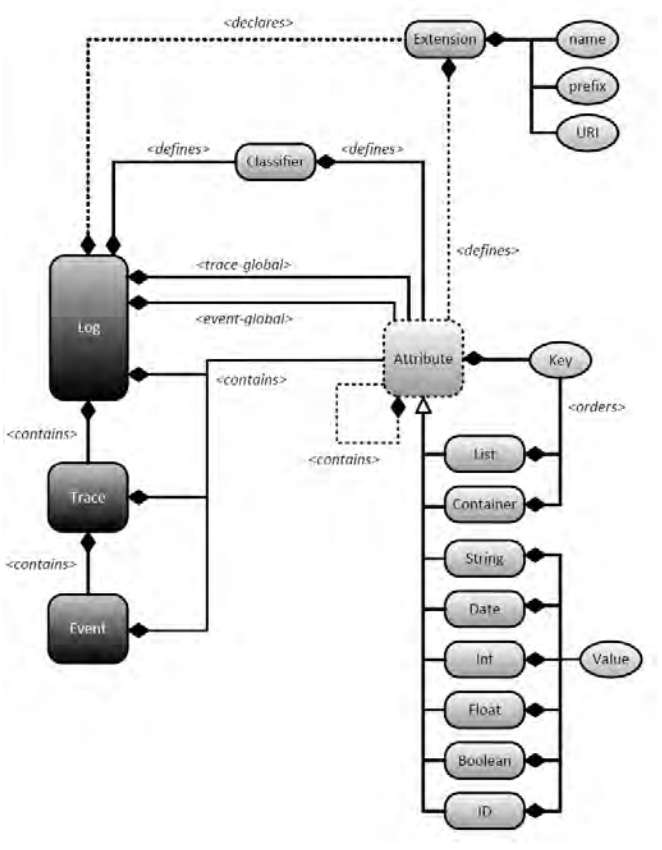


Figure 3.1: The UML 2.0 class diagram for the complete meta-model for the XES standard [15]

Although XES has been established as the standard format for process mining by the *IEEE Task Force on Process Mining* in September 2010, in reality, event logs will not always be in that format. Therefore, process mining tools such as *ProM* offer an import plug-in that can convert different formats such as CSV-files or MS-Access files into an XES conform file. During this thesis, some event logs will be CSV-files, and therefore this plug-in will be used to generate an XES conform event log before applying process mining algorithms.

3.2 Nets

One of the goals of process mining is to extract a process model from an event log. To be able to get a deeper understanding of process mining, it is essential to first understand the underlying concepts of both Petri nets and WorkFlow nets. There are a multitude of other modeling languages, but most process mining algorithms use Petri nets to represent the process model [17]. To be more precise, a subclass of Petri nets called WorkFlow nets [18] is used because the additional characteristics suit the characteristics of business processes. It is easy to translate a Petri or WorkFlow net into other modeling languages like *BPMN 2.0* or *EPC*. An overview of different process modeling languages can be found in Section 3.2 of [14]. Moreover, Petri and WorkFlow nets are a well-established concept and many different characteristics can be proven on them. Proving such characteristics on other process modeling notations may be much more difficult. The first section introduces the concept of Petri nets, including the different node types and the firing rules. The following section introduces the concept of Extended Petri nets, and the last section introduces WorkFlow nets.

3.2.1 Petri nets

Petri nets originate from the dissertation of Carl Adam Petri back in 1962 [19]. Since then a plethora of different use cases have been identified for both academic and industrial purposes. For a more detailed overview about the development of Petri nets please be referred to [20].

A Petri net is a directed bipartite graph that consists of two different node types called places and transitions. Places are represented by circles and transitions by rectangles. The connection between places and transitions is represented by flow relations, and connections between two similar node types are not allowed. In this thesis, the flow relations are denoted as arcs or edges, and the expressions are used as synonyms.

There exist some extended definitions of Petri nets with weight functions and an initial marking, but the provided definition is most commonly used in the context of process mining [14]. A definition of Extended Petri nets can be found in Section 3.2.2.

Definition 1. *Petri Net [19, 14, 21]*

A Petri net is a triplet N =(P, T, F) where:

1. *P =* $\{p_1, p_2, ..., p_m\}$ *is a finite set of places,*
2. *T =* $\{t_1, t_2, ..., t_n\}$ *is a finite set of transitions such that* $P \cap T = \emptyset$ *and*
3. *F* $\subseteq (P \times T) \cup (T \times P)$ *is a set of directed arcs called flow relation or edge*

In Figure 3.2 an example Petri net is shown. The displayed Petri net consists of three different places, denoted as p_1, p_2, p_3, two transitions denoted as t_1 and t_2, as well as six edges. The edges can be identified by the places and transitions they connect. As a result, the edges are $(p1, t1), (p1, t2), (p2, t1), (p2, t2), (t1, p3)$ and $(t2, p3)$.

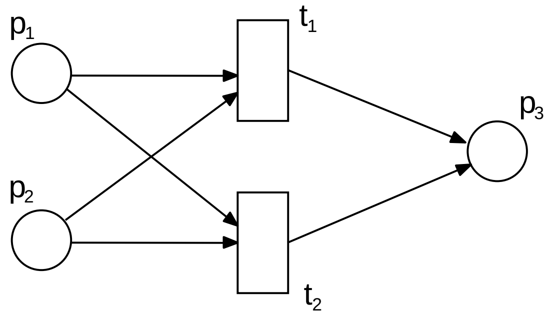


Figure 3.2: An example Petri net with two transitions, three places and six edges

The biggest advantage of Petri nets is its ability to represent dynamic behavior by the so-called firing rule. In the context of process mining, the different states are able to represent various stages of the process execution. Moreover, a Petri net can represent different behavior such as concurrency or choice which are also commonly used in processes.

Firing Rule for Petri nets

Petri nets can describe behavior by states and their changes. One or more than one token represents the current marking of the net. To simulate dynamic behavior a state or a marking within a Petri net is changed according to the firing rules. Before formalizing the firing rules the notation of input and output places needs to be defined.

Definition 2. *Input and output places [14]*

Let N = (P,T,F) be a Petri net. Elements of $P \cup T$ are called nodes.

1. *A place x is an input place of a transition y if and only if there is a flow relation from x to y (i.e., $(x, y) \in F$).*
2. *A place x is an output place of a transition y if and only if there is a flow relation from y to x (i.e., $(y, x) \in F$).*
3. *For any node $x \in P \cup T, \bullet x = \{y \mid (y, x) \in F\}$ and $x\bullet = \{y \mid (x, y) \in F\}$*

Recalling Figure 3.2, p_1 and p_2 are input places for t_1 and t_2 and p_3 is their output place. After introducing the concept of input and output places, the firing rules can now be defined.

Definition 3. *Firing Rules for Petri Nets [22]*

Let (N, M) be a marked Petri net with N = (P,T,F).

1. *A Transition $t \in T$ is enabled if and only if each input place contains at least one token.*
2. *An enabled transition may or may not fire (depending on whether or not the event actually takes place).*
3. *Firing an enabled transition t removes one token from each input place p of t, and adds one token to each output place p of t.*

In other words, a transition may only fire if there is at least one token in each of its input places. When firing, a token is consumed from each of the input places of the transition and a token is produced in each of its output places [20].

3.2.2 Extended Petri nets

Petri nets are a very simple representation and therefore not able to express all constructs that are present in real-life workflows. As a result, some extensions have been made to provide either more convenience (e.g. by adding arc weights) or increase expressiveness (e.g. reset and inhibitor arcs).

Definition 4. *Extended Petri net [23]*

An extended Petri net is a tuple (P,T,F,W,A,L,R,H), where:

1. *Let (P,T,F) be a basic Petri net,*
2. *W:F $\rightarrow$ N be a weight function*
3. *A is a set of (activity) labels,*
4. *L $\in$ T $\rightarrow$ A $\cup\{\tau\}$ is a labeling function*
5. *R $\in$ T $\rightarrow$ 2^P is a function defining reset arcs, and*
6. *H $\in$ T $\rightarrow$ 2^P is a function defining inhibitor arcs.*

A weight function W:F $\rightarrow$ N can allocate weights to edges which allows modeling more behavior, in a sense that a transition can now consume more than one token from an input place. In addition, a transition can only fire if all input places contain enough token. If the weight of the edge is set to 2, then the input place needs to contain 2 tokens for the transition to fire.

Reset arcs R $\in$ T $\rightarrow$ 2^P deal with cancellation behavior. It connects a place with a transition, with the semantic of removing all tokens from the place once the transition fires [24].

Similar to traditional edges, inhibitor arcs H $\in$ T $\rightarrow$ 2^P also go from a place to a transition. Regarding their behavior, inhibitor arcs can be seen as the inverse of a normal arc. The transition is not enabled if the number of tokens in the place is at least as high as the weight of the inhibitor arc. In the default case, the transition may only fire if the place that is connected via an inhibitor arc is empty. Graphically, inhibitor arcs are represented as edges with a circle at the transition [25].

3.2.3 WorkFlow nets

Event logs used for process mining represent different instances of the same process which has been executed multiple times. In the context of a questionnaire, this could represent different participants filling in a similar questionnaire independently. A questionnaire usually has a clear starting and a clear ending point, e.g. the first and last question. Similar to this, business processes usually also have one start and end activity. A WorkFlow net is a subclass of Petri nets, with two special places: the source place i and the sink place o. The source place represents the begin of a process and the sink place represents the end of it [22]. This characteristic of WorkFlow nets matches the behavior observed by questionnaires and business processes.

Definition 5. *WorkFlow net [23]*

An Extended Petri net PN = (P,T,F,W,A,L,R,H) is a WorkFlow net if and only if

1. *There is a single source place i, i.e., $\{p \in P | \bullet p = \varnothing\} = \{i\}$*
2. *There is a single sink place o, i.e., $\{p \in P | \bullet p = \varnothing\} = \{o\}$*
3. *Every node is on a path from i to o, i.e. for any $n \in P \cup T : (i, n) \in F^{*} and (n, o) \in F^{*}$ where F^{*} is the reflexive transitive closure of relation F.*
4. *There is no reset arc connected to the sink place, i.e. $\forall_{t \in T} o \notin R(t)$.*

The same definition can also be applied to normal Petri nets (PN = (P,T,F)) with the simple adaptation that there are no reset arcs in the first place with makes point four obsolete. This definition can be found in [14].

An example WorkFlow net is shown in Figure 3.3. This WorkFlow net consists of nine different places (P1-P9), eight different transitions (T1-T8) and 18 flow relations. Place

P1 represents the source place i, while place P9 represents the sink place o. Moreover, P1 contains a token. This token, represented by a black dot, enables transition T1. Firing transition T1 results in the token from P1 being consumed, and a new token being produced in place P2. The token in P2 does then enable transition T2. After firing T2, both transitions T3 and T3 are enabled, but only one token may be consumed. As a result, a choice between firing transition T3 or T4 has to be made. This represents the choice between two activities in a process of which only one can be executed. Moreover, similar to Petri nets, WorkFlow nets also allow modeling concurrency. This is displayed in Figure 3.3 after firing transition T5. A token is produced in both places P5 and P6, which enables both transitions T6 and T7.

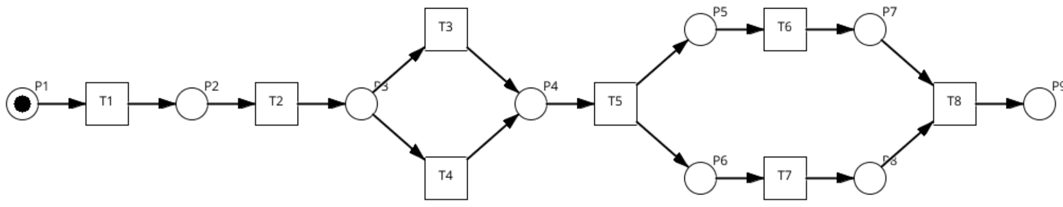


Figure 3.3: A Workflow net with nine places (P1 - P9) and eight transitions (T1 - T9) and a token in P1.

3.3 Decision Trees

In the context of data mining, decision trees are a well-developed approach for classification [26, 27]. In Figure 3.4, an example decision tree is provided that classifies fruits. Please note that this is not a general representation of nodes and branches and that different representations exist. A decision tree consists of a root node, inner nodes, branches, and leaf nodes.

Inner nodes denote a test on an attribute and are represented with red borders in Figure 3.4. An example of an inner node is the *"Taste?"*-node. The root node is a special type of inner node and serves as the starting point of the decision tree. As a result, it is located at the top of the tree. In Figure 3.4 the root node is the *"Color?"*-node.

Branches represent the possible outcomes of a test on an inner node. In Figure 3.4 the branches are represented as grey labels on the lines connecting nodes. An example is the label *yellow* connecting root node with the inner node *"Shape?"*. The last element of decision trees are leaf nodes, which indicate a class of the object. In Figure 3.4 leaf nodes are indicated by green borders. A lot of algorithms that are capable of constructing a decision tree have been developed, such as the ID3 algorithm [28] or its extension the C4.5 algorithm [29].

In the context of process mining, a decision tree can be seen as a visual representation of a set of disjoint decision rules leading to a certain decision within the process instance. These rules can again be used as a classifier for the given data set. Based on an observed set of attribute values, each decision rule can predict a target class [30]. A target class then represents the decision made at the respective decision point during process execution.

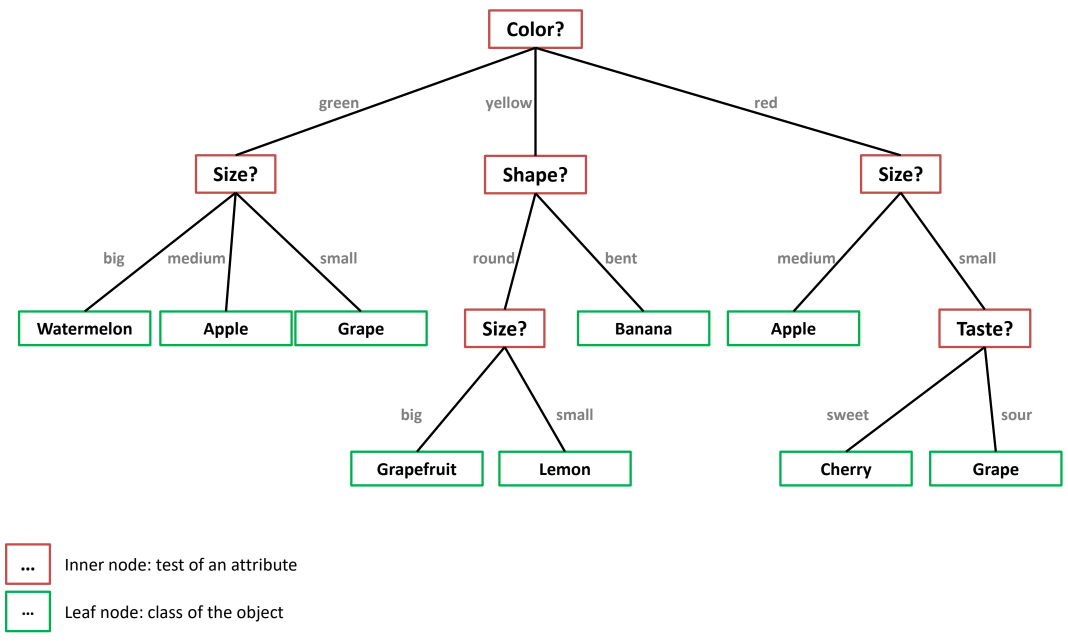


Figure 3.4: Decision tree to classify fruits [31]

3.4 QuestionSys

The previous sections focused on the input, see Section 3.1, and output, see Section 3.2, of process mining. The goal of this section is to create an understanding of the various data collection scenarios. A data collection scenario represents a process-oriented questionnaire that has been executed within the *QuestionSys* framework.

The main goal of *QuestionSys* is to enable people without programming skills to develop data collection instruments. Furthermore, they should be empowered to deploy and execute these instruments on mobile devices. Additionally, the cost of data collection should be decreased by reducing development time and cost, while also increasing the data quality [32]. *QuestionSys* is a generic and flexible questionnaire system which enables process-driven mobile data collection. Mapping the questionnaire to a process model allows its execution by a lightweight process engine, even on mobile devices [2]. Due to the mapping of the questionnaire to a process model, process mining algorithms may be applied to event logs generated during the execution.

The first subsection briefly explains the problem solved by *QuestionSys*, the second subsection describes the requirements towards the system. The last subsection explains the different kinds of event logs generated by *QuestionSys*.

3.4.1 Problem definition

Collecting data from paper-based questionnaires is a very cumbersome and error-prone task due to people having to copy information by hand. Errors may occur when filling in the questionnaire, transferring the data for analysis or during evaluation. With the help of *QuestionSys*, these errors can be reduced and therefore data can be collected both more conveniently and with a higher quality compared to paper-based approaches. Furthermore, a large amount of data may be collected in a rather short time period [3]. In addition, data such as the exact time needed to answer a questionnaire or a specific question can be collected with a higher precision compared to paper-based approaches. Moreover, new data can be collected on the behavior of participants during the questionnaire.

Examples of such data may be the number of answer changes within a specific question or the exact moment in which the participant dropped out of the questionnaire. This data may then be used to get more sophisticated insights into the results, but also improve the structure of the questionnaire itself.

3.4.2 Requirements

Based on different case studies, expert interviews and literature analyses regarding the implementation of mobile data collection applications [33, 34] five different requirements for the mobile support of electronic questionnaires have been defined in [35].

1. *Mobility*

   The data collecting process usually requires extensive interactions. In many situations, computers are disturbing when filling out a questionnaire. To enable more convenient and flexible data collection, the device needs to be portable.

2. *Multi-User support*

   Because different users may interact with the same device or questionnaire, multi-user support is crucial. Additionally, it must be possible to differentiate between different user roles such as interviewer or subject, and users should be able to possess multiple roles.

3. *Support of Different Questionnaire Modes*

   In general, a questionnaire may be used in two different modes: interview and self-rating. Based on the mode, a questionnaire may diverge in the order the questions are posed, the possible answers or additional features. As a result, mobile questionnaire applications should be able to support both modes.

4. *Multi-Language Support*

   Since actors may understand different languages, the person accessing the questionnaire should be able to choose his preferred language. This increases the number of possible participants.

5. *Maintainability*

   Questionnaires may change over time. As a result, it should be possible to quickly change both structure and context of the questionnaire without the need of programming skills.

These requirements are also important in regard to process mining. The first requirement allows answering a questionnaire with a mobile device, e.g. a smartphone or a tablet. Sensors within smartphones and tables can be used to collect additional information about the environment in which the questionnaire is answered [36].

The requirement of Multi-Language Support is also important in the context of process mining. As described in guideline three in Section 3.1.3, references should be stable. In other words, references should be independent from the language setting of the questionnaire. To be able to analyze questionnaire instances collected in different languages, it is important to either translate the answers beforehand or use language independent identifiers for the representation of each question and answer in the event log.

3.4.3 Generated Logs

As briefly described in previous sections, more data may be generated from a questionnaire with *QuestionSys* compared to its paper-based equivalent.

When answering a questionnaire data is collected and documented within event logs. In the context of *QuestionSys*, different types of event logs are generated:

1. Processlog

   This event log documents at what time a specific node is started, processed, executed and destroyed.

2. Resultslog

   This is a digital representation of the completed paper-based questionnaire. The fact that the recorded data is in a digital format from the get-go not only reduces media disruption but also prevents errors when copying data into a digital system.

3. Historylog

   In comparison to the second event log, this one also contains all answers given, even if they have not been submitted or changed. It is not possible to record this kind of data from paper-based questionnaires. As a result, there is no well-established way of analyzing such data.

These different logs are used as input when creating the synthetic event logs used for process mining within this master's thesis.

Note that the process engine of *QuestionSys* does not allow for any deviations from the specified process model [2, 37]. As a result, all event logs used in this thesis can be considered both complete and noise free. Nevertheless, different algorithms are presented that are also able to deal with both noise and incompleteness of event logs.

Figure 3.5 shows the mapping of questionnaire elements to the elements of a process model and event log. This mapping is used in the context of this thesis, to better analyze each question. Using this mapping allows discovering each question instead of each questionnaire page. It is also possible to map the questionnaire pages to the event log event, but this would then restrict the analysis to the page level instead of each question. Moreover, some process discovery algorithms like the Fuzzy Miner are able to cluster activities in the process model which can be used to represent the pages.

Additional data, such as timestamps or answers given, are represented as data attributes of the corresponding event.

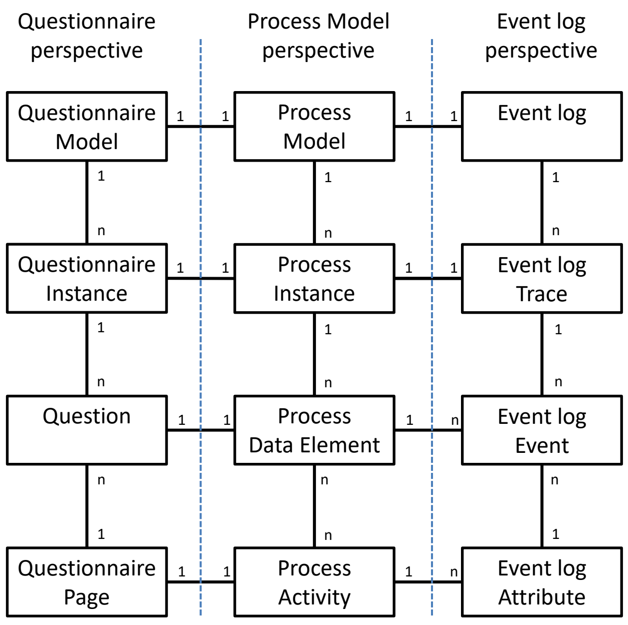


Figure 3.5: Mapping a questionnaire to an event log

3.5 Process Mining Tools

To be able to apply process mining algorithms successfully, tools supporting these algorithms are necessary. In the academic context, the lion's share of research in the area of process mining is conducted by using and also extending the Process Mining framework (*ProM*). Many different commercial tools such as *Disco* or *Celonis* prove that process mining may also solve problems occurring outside of the academic context. This section will only give a very brief overview of two process mining tools, *ProM* and *Disco*, followed by a short listing of other process mining tools to illustrate the variety of different tools.

The goal of this section is to give a brief overview of the available tools and not to give any recommendation or compare the tools. Every tool certainly has its strengths and weaknesses. For a more detailed overview of available process mining tools, please be referred to Chapter 11 of [14].

3.5.1 ProM

The Process Mining framework (*ProM*) is an Open Source framework developed by the TU Eindhoven which provides a platform for users and developers of process mining algorithms. It can be freely downloaded from www.promtools.org or www.processmining.org. Back in 2002 several simple process mining tools were available [38]. As it did not make sense to build a new tool for each new algorithm, *ProM* was developed as a "plug-able" environment for process mining [39]. In 2004, the first version of *ProM* (*ProM 1.1*) was released, which already contained 29 plug-ins. Over time, more and more plug-ins have been developed and added to *ProM*. The release of *ProM 5.2* in 2009 already contained 286 plug-ins. In 2016, there are over 1500 plug-ins available, including deprecated ones, which is why it is impossible to provide a complete overview of *ProM's* functionalities [14]. With the release of *ProM 6*, the execution of plug-ins can be distributed over multiple computers to increase performance [40]. Unfortunately, not all plug-ins from *ProM 5.2* have been re-implemented yet, so at some point, it might be necessary to use an older version of *ProM*.

The *ProM* version used in this thesis is *ProM 6.7* Revision 35885.

3.5.2 Disco

Disco was developed in 2009 to help organizations to regain control over their processes. Based on process mining consulting projects and interviews with practitioners it became clear that process mining needs to be fast and easy to be applied in practice [41]. Based on these requirements, *Disco* was developed. It allows for a very easy import of different standard log formats such as MXML or XES but also supports many other formats such as CSV or MS-Excel files. The event logs are then used for automated process discovery with a next-generation fuzzy miner algorithm. In comparison to *ProM*, *Disco* is very easy and interactive to use and allows to generate expressive process models without much effort. Additionally, many standard event logs are included in *Disco*, which allows for a fast and easy first take on process mining.

A screenshot of *Disco* is shown in Figure 3.6. The two sliders on the right-hand side allow to adjust the granularity of the activities and paths, while the bottom part of that menu allows changing from the frequency to the performance perspective. Moreover, *Disco* also allows for process statistics, log filtering, as well as performance highlighting and animation. As providing a whole overview of the functionalities of *Disco* would extend the scope of this thesis, please be referred to [41] or https://fluxicon.com/disco/ for further information.

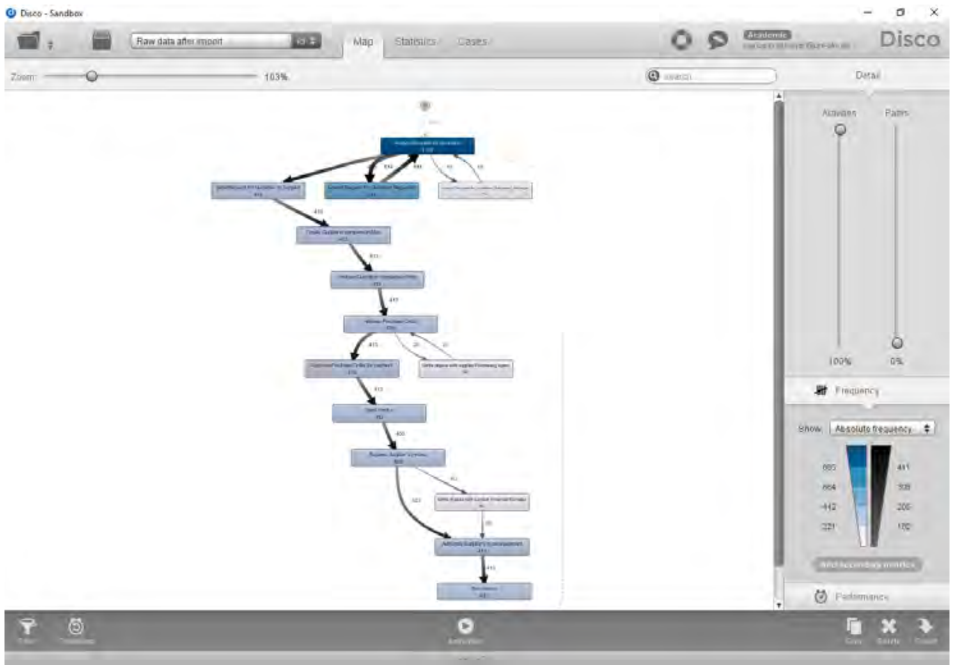


Figure 3.6: A screenshot of the Process Mining Tool Disco

3.5.3 Other Process Mining Tools

There are more tools supporting process mining than just *ProM* and *Disco*. This section should provide a brief overview of other tools.

Process Mining tools may be categorized into academic and commercial tools.

Although the lion's share of academic research is conducted in *ProM*, other academic tools have also been developed. Other academic tools are *PMLAB*, developed by the group of Josep Carmona at the Universitat Politècnica de Catalunya in Barcelona [42] or the benchmarking framework *CoBeFra* which has been developed by the department of Management Informatics at KU Leuven in Belgium [43].

In the context of commercial process mining tools, the variety of tools is greater. The goal is not to give detailed information or compare any tools in this section, which is why the tools are only named in an alphabetical order. Furthermore, these tools are constantly improving with each new release, which makes a case-specific selection of the correct tool necessary. Commercial process mining tools, ordered alphabetically, are *Celonis Process Mining*, *Disco*, *Enterprise Discovery Suite*, *Interstage Business Process Manager Analytics*, *Minit*, *myInvenio*, *Perceptive Process Mining*, *QPR Process Analyzer*, *Rialto Process*, *SNP Business Process Analysis*, and *webMethods Process Performance Manager* [14].

The variety of different commercial process mining tools reflects the industry's great interest in the process mining technology. The fact that process mining algorithms have been implemented in various academic and commercial systems such as the above-mentioned ones proves that there is high interest from both an industrial and academic perspective. More and more software vendors are adding process mining components and functionality to their tools [44].

4 Process Mining

There are some misconceptions related to process mining. Process mining is often reduced to a special data mining technique to discover process models from event logs. Although process mining is able to accomplish this, this only represents a part of its capabilities. Section 4.1 explains why the scope of process mining is not limited to discovery. Process discovery is only one of the three forms of process mining. In fact, process mining may be seen as the connection between data science and process science. Process science combines the knowledge of information technology with management science to execute and improve processes [14], whereas data science uses scientific methods, processes, algorithms, and systems to extract knowledge and insights from data [45].

The main idea of process mining is to generate knowledge from event logs by combining methods from both mentioned disciplines to discover, monitor and improve real processes, which is also displayed in Figure 4.1.

In the context of process mining, data mining techniques such as decision trees are used to identify patterns within event logs (see Section 7.3) and databases are used to both store and provide relevant event data. Furthermore, statistics may be used to characterize event logs and algorithms are used to generate process models from event logs. Methods from predictive analytics may be used to predict the remaining time until a process is completed as described in [46]. Organizational mining (see Section 7.2) can be seen as a method of behavioral science.

Methods from process science such as optimization or business process improvement are directly supported by process mining in a sense that conformance checking directly refers to the detection of deviations between the behavior seen in the event log and the modeled behavior.

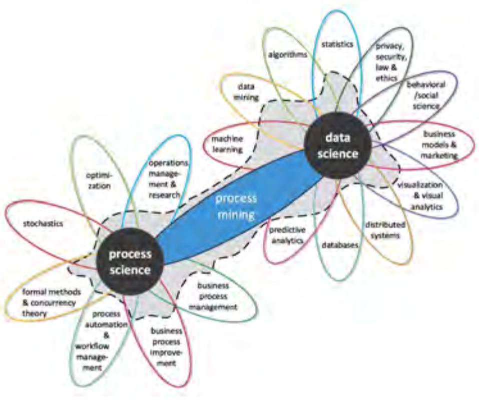

algorithms
statistics
privacy, security, law & ethics
data mining
behavioral /social science
machine learning
data science
business models & marketing
optimi-zation
operations manage-ment & research
process mining
visualization & visual analytics
stochastics
predictive analytics
databases
distributed systems
process science
business process management
formal methods & concurrency theory
process automation & workflow manage-ment
business process improve-ment

Figure 4.1: Positioning of process mining as a bridge between two disciplines [14]

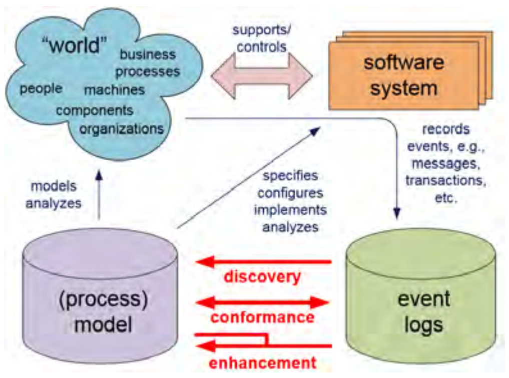


Figure 4.2: Positioning of the three different process mining forms [44]

In addition to the positioning provided in Figure 4.1, process mining may also be positioned in a little less abstract way. Figure 4.2 shows this positioning. Process mining consists of three different forms: discovery, conformance, and enhancement, represented by the red arrows in Figure 4.2. Moreover, Figure 4.2 allows for a more precise positioning of process mining as it builds the connection between a (process) model, which models and analyzes different aspects of reality and event logs recorded from reality via software systems.

These two ways of positioning process mining help to establish a brighter understanding of process mining as a whole before going into more detail in the following chapters.

This chapter aims at creating a better understanding of process mining in general and is structured as follows. Process discovery, conformance checking, and enhancement are briefly introduced as the three forms of process mining in the first section. Section 4.2 introduces the BPM lifecycle and explains in which part of it process mining can be beneficial. The following section then answers the question why process mining may be seen as a nontrivial task and introduces four different competing quality criteria of process mining.

4.1 Three Forms of Process Mining

The term process mining summarizes three different forms of process mining. The first form is *process discovery* where an event log is used to generate a process model. The second form is *conformance checking* in which a process model is compared with an event log of the same process. The third form is *enhancement* where additional information in the event log or deviations identified during conformance checking are used to improve a process model [44].

This section only provides a very brief overview of the three forms, as they are explained and applied in later sections of this thesis. Figure 4.3 describes the three forms of process mining in regard to their input and output.

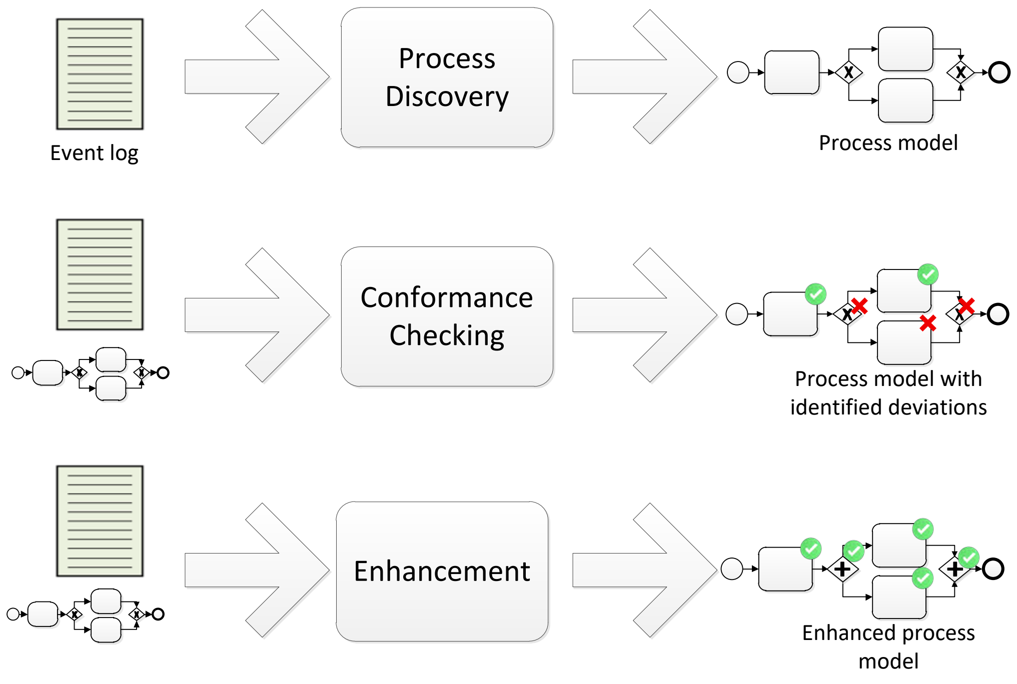


Figure 4.3: Input and Output of Process Mining [44]

4.1.1 Process Discovery

Process discovery is the first and most prominent form of process mining. A process discovery algorithm generates a process model solely based on the information from an event log. Because no other information - such as an already existing process model - is used, it is ensured that only the actual process is discovered. It is surprising for many organizations that process discovery algorithms, which are explained and applied in Chapter 5, are able to discover real processes solely based on an event log [44].

4.1.2 Conformance Checking

The second form of process mining is conformance checking. When checking conformance, a process model is compared with an event log of that process to identify bottlenecks or (un-)wanted deviations. These deviations help to better understand and improve the process. The process model may exist beforehand or is generated by a process discovery algorithm [44]. Conformance checking may therefore not only be used to evaluate a process model but also to evaluate a process discovery algorithm by checking the conformance of the resulting model with the event log used to discover it.

4.1.3 Enhancement

The third form of process mining is enhancement. Enhancement deals with the extension of a process model with additional information or improving the existing model using information about the actual process recorded in some event log. This information, for example, information on the originator or the data generated during the process, is usually not used during the other two forms. Whereas conformance checking measures the alignment between model and reality, this third form of process mining aims at changing or extending an existing process model based on either additional information contained in the event log or deviations identified during conformance checking [44]. This allows to better understand various perspectives such as the data, organizational or temporal perspective of a process.

4.2 Process Mining across the BPM Lifecycle

The BPM lifecycle shown in Figure 4.4 describes six different phases of a business process. During the *Process identification* phase, a process is identified and a process architecture is created.

The following phase is *Process discovery*. This phase uses the process architecture as an input to create an as-is process model that represents the current process as it is executed in reality. It is evident that process mining, especially process discovery, can directly contribute to the discovery of process models. Furthermore, as it only uses event data, process discovery obtains process models that are closer to reality than process models conducted from interviews. If a process model already exists it can be used as the as-is model.

Then the as-is process model is analyzed to get insights into the weak points such as bottlenecks or undesired behavior in the *Process analysis* phase. During this phase, process mining algorithms from conformance checking can directly contribute by revealing deviations within the discovered process model. Additionally, by using data from the event log such as timestamps, more insights into the process performance like bottlenecks can be gained with relatively low effort. These insights are then used in the *Process redesign* phase to create a to-be process model. The to-be process model is often created by a user who is not supported in any way. In the context of process mining, approaches have been done to at least support the user with possible improvements rather than letting him come up with the new process model by himself [47].

In the *Process implementation* phase, the process is implemented to create an executable process model which can then be used. While the improved process model is executed, it will be monitored and controlled in the *Process monitoring and controlling* phase. During this phase, conformance checking algorithms like the Token Replay Algorithm can be used to monitor the process and discover bottlenecks.

In the case of poor process performance or serious demands identified in the previous phase, a new iteration of the BPM lifecycle may be triggered. The next iteration then starts with the redesign phase. Process mining can be used to support all phases of the BPM lifecycle [48, 49].

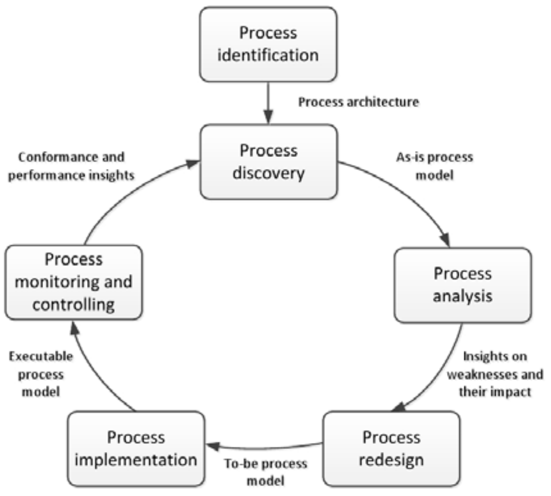


Figure 4.4: The BPM lifecycle [49]

4.3 Why is Process Mining difficult?

When applying process mining algorithms in the context of process discovery, the real process usually is unknown. The data available through event logs is based on the real processes and represents different example executions of the real process. Based on the information contained in the event log, different process discovery algorithms are able to generate a process model. If the event log used to generate a process model only contains a fraction of all possible behavior, the discovered process model can only represent the fraction of behavior observed by the event log.

When applying process discovery algorithms, the main question is whether the process model is a correct representation of the real process. In order to assess whether the discovered process model indeed is a correct reflection of the real process, the four competing quality criteria of *fitness*, *precision*, *generalization*, and *simplicity* will be introduced next.

4.3.1 Four Quality Criteria of Process Mining

Determining the quality of a process discovery result is characterized by many different and competing dimensions, which makes the task of assessing the quality itself very difficult. Some traces may have different probabilities or frequencies or the process model allows for an infinite number of different traces (if it contains loops), which makes the assumption that every possible trace is present in the event log unrealistic. To assess the quality of a process model, the four competing quality dimensions of *fitness*, *precision*, *generalization*, and *simplicity* have been introduced in [44]. Figure 4.5 gives a high-level overview of the four quality criteria.

The four criteria are:

1. *Fitness*

   Fitness refers to the dimension that the discovered model should allow for the behavior seen in the event log. A process model with good fitness allows for the behavior represented in the event log. Perfect fitness may be achieved if all traces from the event log can be replayed by the process model [44]. Different methods to quantify fitness are presented in Chapter 6.

2. *Simplicity*

   The dimension of simplicity refers to the principle of *Occam's Razor* describing that the simplest model that can explain the behavior seen in the event log is the best one. Simplicity may be quantified by the number of nodes and edges in the discovered process model. More sophisticated metrics also taking factors like entropy or structuredness of the process model into account, may also be used to quantify simplicity [14]. For an empirical evaluation of different model complexity metrics, please refer to [50].

3. *Generalization (avoid overfitting)*

   A process model is overfitting in a sense that it is too specific. Since event logs only contain an example behavior of a process, the process model should also allow for the execution of future instances by generalizing the process model [44].

4. *Precision (avoid underfitting)*

   Precision may be seen as the opposite of generalization. While generalization refers to overfitting, precision refers to underfitting. A process model has a low precision or is underfitting if it allows for behavior completely different from the behavior seen in the event log [51].

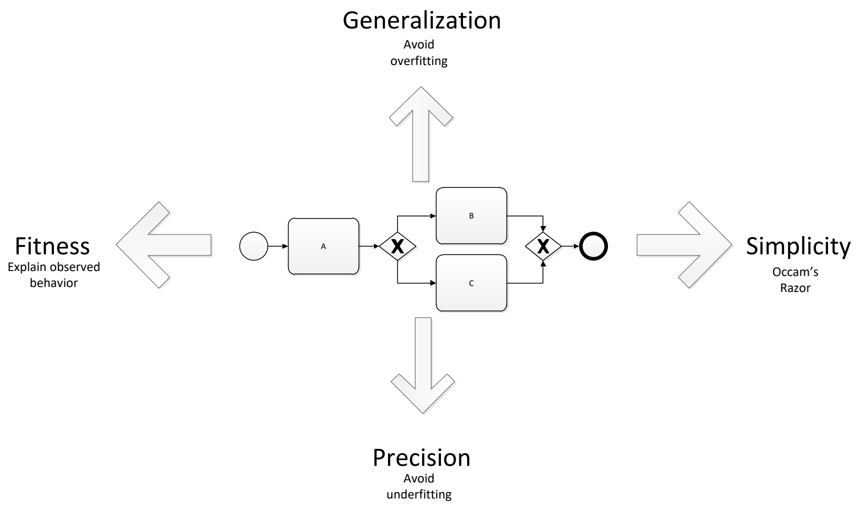


Figure 4.5: Four competing quality criteria for process mining

As shown in Figure 4.5, the different quality criteria described can be interpreted as different forces dragging on a process model. Because the quality criteria are competing, a good trade-off between the different dimensions has to be found. If the process model has a high fitness, it can replay all traces from the event log, which usually also increases the complexity of the process which then again results in low simplicity. A similar trade-off can be observed between generalization and precision. If the process model allows for behavior that is not related to the event log it has good generalization but bad precision and vice versa. Completely ignoring these dimensions during process discovery will lead to degenerate cases.

5 Process Discovery

After introducing and positioning process mining in the previous chapter, a better understanding of process mining in general is created. Process discovery was introduced as the first form of process mining, and this chapter aims to provide a better understanding of the algorithms used in this form. Process discovery is arguably one of the most challenging tasks within the process mining discipline. It refers to constructing a process model solely based on the information captured by an event log. Patterns in the event log are identified and used to derive behavior which is then translated into a process model. This chapter explains different process discovery algorithms and illustrates how they may be used in the context of data collection scenarios. A data collection scenario is represented by a questionnaire performed with *QuestionSys* (see Section 3.4 for more information on *QuestionSys*).

This chapter is structured in the following way. First, the focus is set on the Alpha Algorithm, because it grants an easy introduction to the topic of process discovery. Its extensions then allow discovering interesting properties in regard to questionnaires. The following section introduces the Heuristic Miner which does take frequencies into account when constructing the process model. This characteristic makes this algorithm more robust towards incompleteness and noise in the event log. As the third algorithm, the Fuzzy Miner takes a completely different approach to the discovery of a process model and allows for interesting analysis such as animation. Last, Genetic Process Mining uses the potential of a genetic algorithm to discover process models.

The second section provides a comparison of the explained algorithms, while also providing a direction towards the correct discovery algorithm based on various contexts.

There are other process discovery algorithms that are not covered in this thesis, including Region-Based Mining which always discovers the most precise process model, but is very susceptible to noise, requires completeness of an event log and has limitations regarding short loops [52]. Multi-Phase mining does not require completeness of event logs but also has problems with noise in event logs [53]. The Inductive Miner always discovers sound process models, but its internal representation does not work on Petri nets [54].

5.1 Algorithms

After providing a brief overview of process discovery, this section now covers different process discovery algorithms that can be applied in the context of process mining. In general, process discovery algorithms may be divided into algorithms following a *local* or *global* strategy. Algorithms with a *local strategy* build the process model step by step, based on the optimal local information. An example of a very local algorithm is the Alpha Algorithm introduced in Section 5.1.1. In contrast to algorithms following a *local strategy*, algorithms following a *global strategy* try to find the optimal model based on a one strike search. Genetic Process Mining, as explained in Section 5.1.8 is an algorithm that follows a very global strategy.

Both strategies have advantages and disadvantages. From a computational point of view, local strategies are less complex and therefore require less computational power compared to global strategies. However, combining the optimal local steps may not always result in the optimal process model. Global strategies are more complex from the computational perspective but have a higher chance of finding the optimal solution [55]. A global strategy may take a considerable amount of time and does not guarantee to return the optimal model. These are only the two extreme strategies. In some approaches, local and global strategies are combined in a sense that first a local approach is performed and the result is then checked by a global approach [56].

This section first introduces the basic Alpha Algorithm including its prerequisites and the ordering relations. Then the Alpha Algorithm is defined and the some of its limitations are explained, followed by a brief overview of two extensions, the Alpha+ Algorithm and the Alpha++ Algorithms, which tackle some of the initial problems identified with the basic algorithm. Next, the Heuristic Miner is introduced as an algorithm that also takes frequencies into account when discovering a process model, which makes this algorithm a lot more robust compared to the Alpha Algorithm. In Section 5.1.6 the Fuzzy miner is introduced, followed by the Genetic Process Miner in Section 5.1.8. Additionally, for each algorithm, its applicability in the context of questionnaires is evaluated.

5.1.1 The Alpha Algorithm

This section introduces the Alpha Algorithm [57]. The basic Alpha Algorithm can be used as a good introduction to process discovery because it is very easy to understand and many of its ideas have been included in more advanced process discovery techniques [57, 58]. This subsection explains the Alpha Algorithm, its prerequisites, the ordering relations, the algorithm itself and some of the limitations associated with the Alpha Algorithm.

Prerequisites

The basic idea of the Alpha Algorithm is to scan the event log for special patterns such as *Direct succession, Causality, Parallelism or Choice*. To do so, three prerequisites need to hold regarding the event log [57].

1. each event refers to a well-defined step in the process i.e. a task or an activity
2. each event refers to an instance, i.e. a case
3. events are recorded sequentially, i.e. events are totally ordered

These prerequisites directly correspond to the fundamentals of event logs explained in Section 3.1. If the three prerequisites are fulfilled, the Alpha Algorithm is able to discover a sound WorkFlow net from the event log.

Log-based ordering relations

To construct the process model, the Alpha Algorithm searches for special patterns in the event log. These patterns are identified by using the so-called log-based ordering relations [57]. Different relations like *Direct succession, Causality, Parallelism, and Choice* are used to distinguish between different behavior in the process model.

Definition 6. *Ordering relations $>$, $\rightarrow$, $\#$, and $||$ [57]*

Let W be an event log with task set T and $a, b \in T$, T be a set of executable traces within W and $\sigma \in T^*$ be the representation of the execution of a particular case:*

1. $a > b \Leftrightarrow \exists$ *trace* $\sigma =< t_1, t_2, t_3, \ldots t_{n-1} >$ *with* $\sigma \in T^*, t_i = a \wedge t_{i+1} = b, i \in \{1, 2, \ldots, n-2\}$
2. $a \rightarrow b \Leftrightarrow a > b \wedge \neg(b > a)$
3. $a \# b \Leftrightarrow \neg(a > b) \wedge \neg(b > a)$
4. $a || b \Leftrightarrow a > b \wedge b > a$

The relation $a > b$ is called direct succession and is used to capture the pattern in which the tasks `a` and `b` co-occur, and `a` directly precedes `b`. Furthermore, the direct succession is used to distinguish the other three patterns.

The relation $a \rightarrow b$ contains all pairs of activities in a causal relationship. Causality is characterized if task `a` is followed by task `b`, but `b` is never followed by `a`.

Choice, denoted as $a\#b$, represents that task `a` is never followed by `b`, and `b` is never followed by `a`, i.e. there is no trace in the event log in which `b` is followed by `a` and vice versa.

Parallelism, denoted as $a||b$, represents that sometimes task `a` is preceded by `b`, and sometimes `b` is preceded by `a`.

Despite being easy to understand these ordering relations allow for the discovery of process models with a considerable amount of behavior like AND or XOR constructs. The following section defines the necessary steps of the basic Alpha Algorithm [59].

Definition Alpha Algorithm

This section first defines the basic Alpha Algorithm formally, followed by an explanation of each step [57]. With only eight steps, the Alpha Algorithm is able to discover a sound WorkFlow net from an event log.

Definition 7. *The Alpha Algorithm [17]*

Let W be a event log over T. $\alpha(W)$ *is defined as follows:*

1. $T_W = \{t \in T | \exists_{\sigma \in W} t \in \sigma\}$,
2. $T_I = \{t \in T | \exists_{\sigma \in W} t = first(\sigma)\}$,
3. $T_O = \{t \in T | \exists_{\sigma \in W} t = last(\sigma)\}$,
4. $X_W = \{(A, B) | A \subseteq T_W \wedge B \subseteq T_W \wedge \forall_{a \in A} \forall_{b \in B} a \rightarrow b \wedge \forall_{a_1, a_2 \in A} a_1 \# a_2 \wedge \forall_{b_1, b_2 \in B} b_1 \# b_2\}$,
5. $Y_W = \{(A, B) \in X_W | \forall_{(A', B') \in X_W} A \subseteq A' \wedge B \subseteq B' \Rightarrow (A, B) = (A', B')\}$,
6. $P_W = \{p_{(A,B)} | (A, B) \in Y_W\} \cup \{i_W, o_W\}$,
7. $F_W = \{(a, p_{(A,B)} | (A, B) \in Y_W \wedge a \in A\}$
   $\cup \{(p_{(A,B)}, b) | (A, B) \in Y_w \wedge b \in B\}$
   $\cup \{(i_W, t) | t \in T_I\} \cup \{(t, o_W) | t \in T_O\}$,
8. $\alpha(W) = (P_W, T_W, F_W)$

The Alpha Algorithm applies the eight steps to construct a WorkFlow net (P_W, T_W, F_W) [17].

During the first step, the event log is scanned for its activities. In the second step, the initial transitions are identified. They are the activities that appear in the first position of each trace. Similar to step two, in step three the final transitions are identified. These are the activities that appear in the last position of each trace. After finishing steps one, two and three, all activities, including the first and last one, are identified. Each of this activity is represented as a transition within the resulting net, which is why they are also denoted as T.

Recalling Definition 1 from Section 3.2.1, a Petri or WorkFlow net consists of transitions, places and flow relations. The transitions have been identified, so places and flow relations are yet to be discovered.

Discovering places is done by a number of steps [14]. During step four, two different sets of activities A and B are identified, having the following two properties:

1. Two activities in the set A should never follow one another. Two activities in the set B should never follow one another (including taking the same activity).
2. Any activity from the set A should always be followed by a direct succession from any activity in the set B. So, there should be at least one position in the log where the element of A is followed by the element of B. This should hold for all combinations.

The two sets allow to find out which transitions are causally related [58].

In step five, the set X_W from the previous step is refined by only taking the maximal sets A and B into account. Leaving out step five would possibly result in many unwanted places.

The remaining steps of the Alpha Algorithm are very easy. Step six adds the places from step five to the initial and final place.

At this stage of the Alpha Algorithm, both transitions and places are identified.

Logically, the creation of the flow relations is done in step seven. Each place $p_{(A,B)}$ is connected with each element of its set A and with each element of its set B. Additionally, an edge has to be drawn from the source place to each start transition and an edge from each end transition to the sink place.

Step number eight returns a Petri or WorkFlow net with places P, transitions T and edges F [59].

Limitations of the Alpha Algorithm

The Alpha Algorithm is able to successfully discover a WorkFlow net based on an event log. But it has some limitations when discovering certain behavior, which will be discussed in this section.

To be able to understand the limitations, it is necessary to first understand how the Alpha Algorithm works in a more abstract way.

The behavior of the Alpha Algorithm may be summarized in the following way:

1. Every activity in the event log also exists as a transition in the resulting net
2. A transition has ingoing edges if the activity is the first one in a log trace or the activity causally follows another task
3. A transition has outgoing edges if the activity is the last one within a log trace or the activity is causally followed by another task

As a result, if an activity is neither the first nor the last one in any trace of the event log and additionally it is not involved in any causal relationship, the Alpha Algorithm does not generate edges for this activity [60].

Many different constructs have been identified that may cause problems for the basic Alpha Algorithm. They are briefly explained, but going into more detail would extend the scope of this thesis. If you are interested in a more detailed insight into these limitations, please be referred to [60].

1. *Loops of length one*

   A loop of length one represents the construct that a single task t can be executed multiple times in sequence. This can be represented in a WorkFlow net, if all ingoing places are also the outgoing places of a transition. The Alpha Algorithm requires the causal relation $t \rightarrow t$ to generate a place with the same ingoing and outgoing transitions. To create the $t \rightarrow t$ relation, both $t > t$ and $t \not> t$ need to hold, which is impossible.

2. *Loops of length two*

   In case of a length two loop, the Alpha Algorithm infers parallel activities and therefore no place is created between them. If the relation between the two activities would be $a \rightarrow b$ and $b \rightarrow a$ instead of $a||b$, the Alpha Algorithm would discover the correct WorkFlow net.

3. *Invisible tasks*

   Invisible activities may occur if the activity is not registered in the event log or if there is noise within the event log. Therefore, these tasks do not appear in the event log and consequently, they are not identified during the first step of the Alpha Algorithm. As a result, they are not part of T_W and therefore not represented in the resulting net.

4. *Duplicate tasks*

   It may happen that a task appears more than once within the same model. In this situation, all duplicate tasks are assigned the same label and are also registered in the event log with the same label. The Alpha Algorithm cannot distinguish between different tasks with the same label and is therefore unable to deal with duplicate tasks. To tackle this problem, a heuristic to capture duplicate tasks has to be established.

5. *Implicit places*

   Implicit places have the characteristic that neither their presence nor their absence affects the possible log trace of the workflow. As a result, they do not influence the causal relations between tasks. Since places created by the Alpha Algorithm are based on existing causal relations within the event log, implicit places are not captured. This is also the reason why the Alpha Algorithm is unable to generate *explicit* places between tasks if they do not have a causal relation.

6. *Non-free choice constructs*

   Non-free choice constructs are difficult to mine for the Alpha Algorithm because they represent a combination of synchronization and choice. As a result, processes containing non-free choice constructs are not always mined correctly by the Alpha

Algorithm. If it is possible to create a causal relationship between the activities involved in the non-free choice construct the Alpha Algorithm can create the correct places. If it is not possible to infer this causal relationship, for example, if the non-free choice construct contains a long-distance relationship, then the Alpha Algorithm cannot correctly mine that construct. For further information about non-free choice constructs please be referred to [61].

This section introduced the Alpha Algorithm both formally and in a less abstract way by describing the necessary steps. Moreover, the ordering relations are introduced, and six limitations of the Alpha Algorithm have been discussed. The following section will briefly introduce two extensions of the Alpha Algorithm that have been developed in order to overcome some of the presented limitations.

5.1.2 Extensions of the Alpha Algorithm

As described in the previous section, there exist limitations with the Alpha Algorithm when discovering specific constructs. To tackle these problems, extensions have been developed. The Alpha+ and the Alpha++ Algorithm are two of them and they are briefly explained in this section.

The Alpha+ Algorithm

The Alpha+ Algorithm is the first extension of the Alpha Algorithm. The ordering relation explained in Section 5.1.1 fail to recognize a $a \rightarrow b$ relation if $a > b$ and $b > a$ and both aba and bab are contained in the event log. By introducing a definition of *loop complete event logs* and improving the ordering relation, the basic Alpha Algorithm can successfully mine loops of length two. A proof of this is included in [58].

Loops of length one can produce the substring tt as the two tasks follow each other in the event log. Therefore, these tasks have to be identified together with the single place to which each task is connected. Any length-one-loop task t may be identified by searching for the substring tt within the loop-complete WorkFlow log. The place to which the task should be connected can be identified by checking the transitions that are directly followed by t, and which transitions directly follow t. In other words, the place that is an input place for the following transition and the place that is an output place of the followed transition need to be found. The Alpha+ Algorithm [58] is able to mine both length one and length two loops. Further information on the Alpha+ Algorithm can be found in [58, 62, 63, 64, 65].

A comparison between the mining results of the Alpha Algorithm and its extension the Alpha+ Algorithm is provided in Section 5.1.3

The Alpha++ Algorithm

Based on the advanced functionality provided by the Alpha+ Algorithm [58, 63], the Alpha Algorithm was further extended to deal with so-called non-free choice constructs. An example of a non-free choice construct is provided in Figure 5.4. There is a free choice between activities `A` and `B`, however, this decision influences whether activity `D` or activity `E` is executed later in the process model. The term *free choice* originates from Petri nets. Free-choice Petri nets are a subclass of Petri nets with the restriction that transitions consuming tokens from the same place should have identical input sets [66]. Previous algorithms, like the Alpha and Alpha+ Algorithm, fail to mine specific non-free choice constructs.

In [67], the Alpha++ Algorithm is presented, which is able to discover nets containing non-free choice constructs. The authors identified that the main reason why previous approaches were unable to discover non-free choice constructs is, that possible causal relations between two activities `A` and `B` are only taken into account if the sequence `AB` occurs at least once in the event log. The dependency distance of `AB` is one, whereas the distance of non-free choice constructs is higher than one. By increasing the observed dependency distance with five new ordering relations, this problem is solved by the Alpha++ Algorithm. Further information on the Alpha++ Algorithm, can be found in [68, 67].

5.1.3 The Alpha Algorithm and its extensions in the Context of Data Collection Scenarios

In the context of *QuestionSys*, a questionnaire is translated into a process model. Each answered question may, therefore, be represented as an activity within the process model. Since questions are very different within questionnaires, the process models in this subsection are displayed in a more abstract way. Each activity label in the process models displayed may be replaced by an identifier of the question such as the wording of the question or its ID.

Loops of length one

As described in Section 5.1.2, the basic Alpha Algorithm is unable discover loops of length one. An example is provided in Figure 5.1. The event log contains a loop of length one, which translates into the sequence of `DD` in the event log. As a result, the basic Alpha Algorithm fails to discover the correct behavior of the process model, resulting in an unconnected transition `D` and a transition `C` which has no output places. The described scenario illustrates one of the problems identified in Section 5.1.1 and is displayed in Figure 5.1. The result of applying the Alpha+ Algorithm to the same event log, is provided in Figure 5.2. Note that the Alpha+ Algorithm is capable of correctly discovering loops of length one.

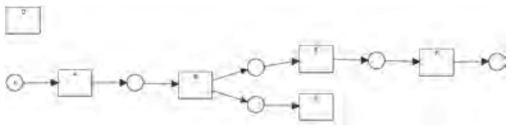


Figure 5.1: Resulting WorkFlow Net from applying the Alpha Algorithm to an event log with length one loop

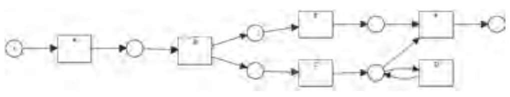


Figure 5.2: Resulting WorkFlow Net from applying the Alpha+ Algorithm to an event log with length one loop

In the context of *QuestionSys* loops of length one are relevant if the same question can be answered multiple times in sequence. In that scenario, the basic Alpha Algorithm is unable to discover the correct questionnaire, whereas the Alpha+ Algorithm is able to which is why the Alpha+ Algorithm would be more suitable for that scenario.

Non-free choice constructs

Figures 5.3 and 5.4 display the results from applying both the Alpha Algorithm and the Alpha++ Algorithm to the same event log. In general, the questionnaire represented in Figures 5.3 and 5.4 has the following behavior: First question `Q` is answered, followed by a choice between questions `A` and `B`. Next, both, questions `C` and `X` are answered, followed by a choice between questions `D` and `E`.

The difference between the two discovered models is, that Figure 5.4 contains a non-free choice construct. The choice between questions `D` and `E` is non-free. If question `A` was answered in the questionnaire, then only question `D` may be answered later, whereas if question `B` is answered, only question `E` may be answered. In other words, the decision between question `D` and `E` depends on the first decision between question `A` and `B`.

Both algorithms discover a correct WorkFlow net, however, the net generated by the basic Alpha Algorithm has a lower precision than the net generated by the Alpha++ Algorithm because it doesn't represent the non-free choice construct.

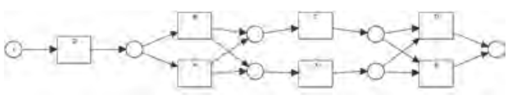


Figure 5.3: Resulting WorkFlow Net from applying the Alpha Algorithm

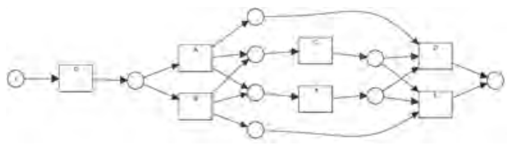


Figure 5.4: Resulting WorkFlow Net from applying the Alpha++ Algorithm

Non-free choice constructs represent a structural characteristic of a Petri net, and as a result they can be identified on a structural level, and may occur in the event log, although they are not explicitly modeled in the questionnaire. If a non-free choice construct is identified, valuable information may be generated for analyzing the questionnaire and its results. Non-free choice constructs indicate that there is in fact a relation between two choices, in a sense that they influence each other. Let's assume the questionnaire is about employee satisfaction, and there is a non-free choice construct indicating there is a relationship between free beverages and the working atmosphere within the department. For example, if free beverages are provided (question `A` is answered) the atmosphere within the department is evaluated as good (question `D` is answered). Although this example may be trivial from a logical perspective, the basic Alpha Algorithm does not allow for such analysis. Because the Alpha++ Algorithm is able to discover non-free choice constructs, analyzing the structure of a Petri net generated by the Alpha++ Algorithm may lead to new and unknown correlations within a questionnaire. Moreover, these correlations may be used to generate novel knowledge about the topic covered by the questionnaire.

5.1.4 The Heuristic Miner

As described in the previous section, the Alpha Algorithm has limitations regarding short loops and non-free choice constructs. When revisiting the behavior of the Alpha Algorithm, described in Section 5.1.1, an edge is created between a place and a transition if the corresponding sequence appears in the event log. This is also the case if the sequence appears only once in tens of thousands of instances.

Representing such infrequent behavior often results in process models that are difficult to understand, and consequently, the Heuristic Miner was developed as a new algorithm that also takes frequencies into account. This allows to leave out infrequent behavior in the resulting model. The Heuristic Miner generates Causal nets (C-nets) compared to the WorkFlow nets generated by the basic Alpha Algorithm. Note that within newer versions of *ProM*, the Flexible Heuristics Miner [69] is implemented which generates a heuristics net or dependency graph instead of a C-net.

The first section briefly introduces the concept of C-nets. Next, the different steps taken to discover a process model with the Heuristic Miner are explained. First, the ordering relations are extended. Based on the ordering relations, a dependency matrix is generated which is then used in combination with various thresholds to derive a process model. Then a way of differentiating between AND & XOR constructs is presented, followed by an approach on how to identify non-free choice constructs. The *ProM* plug-in used in this section is called *Mine for a Heuristics Net using Heuristics Miner*.

Causal nets

Causal nets are a process representation tailored towards process mining in which nodes represent activities and edges represent causal dependencies. Moreover, each activity has a set of possible input and output bindings. Within C-nets, behavior may be represented as described in Figure 5.5.

For a more detailed explanation of C-nets, please be referred to [70].

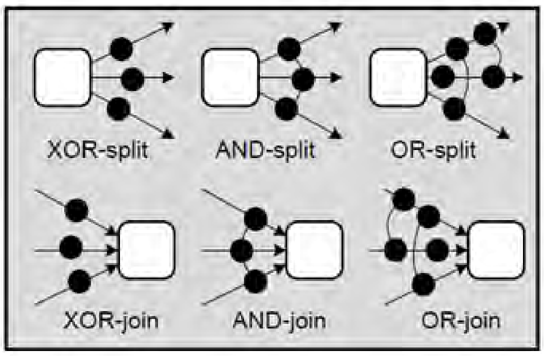


Figure 5.5: Representation of different behavior in the context of C-nets [14, 70]

Extending the dependency relations

After explaining the output generated by the basic Heuristic Miner, the necessary steps to construct a process model are explained in this section. As a first step, the ordering relations from the Alpha Algorithm are extended.

Definition 8. *Extending the Ordering relations [71]*

Let W be an event log over T with $W \subseteq T^*$ and let $a, b \in T$:

1. *$a >_W b$ if and only if there is a trace $\sigma = t_1, t_2, t_3, ...t_n$ and $i \in \{1, ..., n-1\}$ such that $\sigma \in W$ and $t_i = a$ and $t_{i+1} = b$*
2. *$a \rightarrow_W b$ if and only if $a >_W b$ and $b \not>_W a$*
3. *$a \# b$ if and only if $a \not>_W b$ and $b \not>_W a$*

4. $a||_W b$ *if and only if* $a >_W b$ *and* $b >_W a$

5. $a >>_W b$ *if and only if there is a trace* $\sigma = t_1, t_2, t_3, ...t_n$ *and* $i \in \{1, ..., n-2\}$ *such that* $\sigma \in W$ *and* $t_i = a$ *and* $t_{i+1} = b$ *and* $t_{i+2} = a$

6. $a >>>_W b$ *if and only if there is a trace* $\sigma = t_1, t_2, t_3, ...t_n$ *and* $i < j$ *and* $i, j \in \{1, ...n\}$ *such that* $\sigma \in W$ *and* $t_i = a$ *and* $t_j = b$

The first four relations are similar to the relations defined for the basic Alpha Algorithm in Section 5.1.1 [72]. Let's suppose an event log in which the $a >_W b$ relation holds for 999 traces but in one single trace, maybe due to incorrect logging or human failure $b >_W a$ occurs. If this event log is analyzed with the Alpha Algorithm, which does not take frequencies into account, this single outlier would lead to an *"incorrect"* process model because the relationship between a and b would be assumed to be parallel instead of causal. The heuristic mining algorithm, on the other hand, is able to detect that $b >_W a$ is an outlier and should therefore not be taken into account. As a result, the Heuristic Miner is not as sensitive to both incompleteness and noise compared to the Alpha Algorithm. The general idea of the heuristic mining algorithm is to also take frequencies into account when constructing the process model from the relations $a \rightarrow_W b$, $a\#b$ and $a||_W b$. The additional relation of $a >>_W b$ is used to deal with short (length one) loops and the $a >>>_W b$ relation allows to identify long-distance relations.

Three different steps have to be taken in order to apply the heuristic mining algorithm. They will be introduced next.

Step 1: Mining of the dependency graph

The first step of the heuristic mining algorithm is the construction of a dependency graph which indicates how certain it is that there truly is a dependency relation between two events A and B.

A dependency from A to B is denoted as $A \Rightarrow_W B$, and the dependency values between two events are then used to find the correct dependency relations.

Definition 9. *Frequency based metrics [14, 69]*

Let W be an event log over T, and $a, b \in T$. *Then* $|a >_W b|$ *is the number of times* $a >_W b$ *occurs in* W*, and*

$$|a \Rightarrow_W b| = \begin{cases} \frac{|a>_W b| - |b>_W a|}{|a>_W b| + |b>_W a| + 1} & \textit{if } a \neq b \\ \frac{|a>_W a|}{|a>_W a| + 1} & \textit{if } a = b \end{cases} \tag{5.1}$$

The value of $A \Rightarrow_W B$ is always between -1 and 1, and a high value of $A \Rightarrow_W B$ results from a high number of $A >_W B$ and a low number of $B >_W A$ which indicates that there is a high likelihood of a relation in which A is followed by B. The other way around, a low value of $A \Rightarrow_W B$ suggests that there is no dependency between A and B. However, setting thresholds for when a value is high or low is very difficult because the threshold appears to be very sensitive towards the underlying process.

Each non-trivial activity must have at least one dependent activity. This information in combination with the so-called all-connected heuristic allows for an identification of the best candidate. The best candidate is the one candidate with the highest $A \Rightarrow_W B$ score.

Using the same event log, $L =< QACXD, QBXCE, QBCXE, QAXCD >$, as for the Alpha Algorithm previously, this results in the dependency matrix displayed in Table 5.1.

$\|\Rightarrow_W\|$	A	B	C	D	E	Q	X
A	0	0	0.5	0	0	-0.66	0.5
B	0	0	0.5	0	0	-0.66	0.5
C	-0.5	-0.5	0	0.5	0.5	0	0
D	0	0	-0.5	0	0	0	-0.5
E	0	0	-0.5	0	0	0	-0.5
Q	0.66	0.66	0	0	0	0	0
X	-0.5	-0.5	0	0.5	0.5	0	0

Table 5.1: An example of a dependency matrix

Based on the dependency matrix, the first and last activity can be identified. The first activity is the one with no positive values within its column, in the provided example activity Q. The last activity is the one with no negative values within its column. In the provided example there are two final nodes, D and E. To find the next node, the highest value within each activities row has to be identified. The identified first activity can be used as a start. In the example provided in Table 5.1, Q has two subsequent activities, A and B, which are again followed by C and X. Vice versa, the highest value in each column refers to the predecessor of this column's activity. From a matrix similar to the one displayed in Table 5.1, a dependency graph can be derived. An example of such a graph can be seen in Figures 5.6 and 5.7. The number contained in the nodes indicates how often each activity has been executed, whereas the labels on the edges indicate how frequent a specific edge was taken in Figure 5.6 or the dependency in Figure 5.7. For a formal representation of the construction of such a dependency graph, please be referred to [69].

In many cases, it is not clear whether a trace documented in the event log is a low-frequency pattern in the process or noise. Therefore, the Heuristic Miner offers different threshold parameters that indicate which dependencies are accepted:

1. *Dependency threshold,* which allows to restrict the dependencies with a value less than the specified threshold,
2. *Positive observations threshold*, which accepts the dependencies with a frequency higher than the specified threshold,
3. *Relative to best threshold*, which accepts the dependencies with a lower difference towards the "best" dependency than the value of relative to best threshold.

Thresholds like the ones described above help to identify and distinguish between low-frequency activities and noise [71]. Moreover, different thresholds may lead to different process models. The thresholds used to discover Figures 5.6 and 5.7 are: relative to best: 5.0, Dependency: 49.85, length-one-loops: 90.0, length-two-loops: 90.0 and long distance: 90.0.

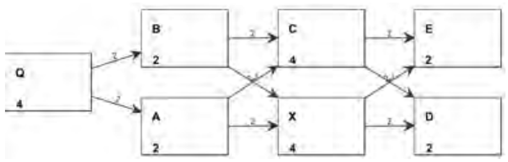


Figure 5.6: Example of a heuristics net displaying frequency on the edges

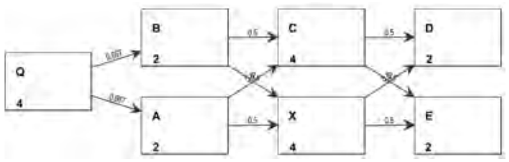


Figure 5.7: Example of a heuristics net displaying dependencies on the edges

Step 2: Dealing with AND/XOR-split/join and non-observable tasks

As seen in Figures 5.6 and 5.7, there are no explicit Splits and Joins within the dependency graph. Additionally, identifying so-called non-observable tasks, in other words, activities which are not represented in the event log, is very difficult. Translating a heuristics net into a Petri net and vice versa is very straightforward and is explained in detail in [73].

The idea behind the mining of logical expressions with the heuristic miner is fairly simple. The dependency matrix and the dependency graph provide information about which activities are input and output expressions of each activity. In an event log, the patterns `...AB...` and `...BA...` may appear if `A` and `B` are in a AND relation, but not if they are in a XOR relation. If they are in a XOR relation, they are not allowed within the same event log. This idea can be formulated as follows:

Definition 10. *Differentiating between choice and concurrency*

Let W be the event log over T with $a, b, c, \in T$ *and let* b *and* c *be in a depending relation with* a. *This allows for:*

$$a \Rightarrow_W b \wedge c \quad = \quad \left(\frac{|b >_W c| + |c >_W b|}{|a >_W b| + |a >_W c| + 1} \right) \tag{5.2}$$

In this formula $|b >_W c| + |c >_W b|$ represents the number of times in which b and c appear directly after each other and $|a >_W b| + |a >_W c|$ represents the number of positive observations. A high value of $a \Rightarrow_W b \wedge c$ indicates an AND-relation while a low value indicates an XOR-relation[71].

Step 3: Mining long distance dependencies - non-free-choice constructs

A non-free choice construct is a combination of choice and synchronization and can be seen in Figure 5.4. In this example, the decision whether transition D or E fires after executing both C and X depends on the first decision between transitions A and B [65]. Mining this kind of behavior is very difficult because the algorithm has to "remember" the decisions made earlier within the process instance. However, as described in Section 5.1.4, the dependency $a >>>_W b$ indicates that kind of behavior[71].

Figure 5.8 displays the result of applying the Heuristic Miner to an event log containing a non-free choice construct. The thresholds used are Relative-to-best: 5.0, Dependency: 50.0, Length one and two loops: 90 and Long distance: 50.0.

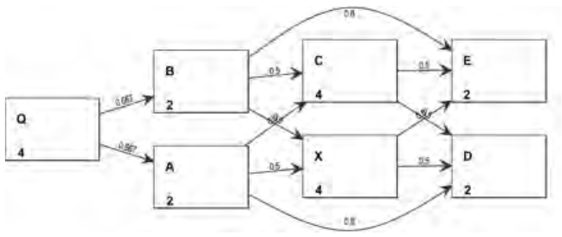


Figure 5.8: Discovering non-free choice constructs with the Heuristic Miner

5.1.5 The Heuristic Miner in the Context of Data Collection Scenarios

The *ProM* plugin *Mine for a Heuristics Net using Heuristics Miner* was used to generate the results displayed in Figures 5.6, 5.7 and 5.8. They show the result of applying the Heuristic Mining algorithm to the same event log as in Section 5.1.3.

The main benefit of using the Heuristic Miner over the Alpha Algorithm in the context of data collection scenarios is that the Heuristic Miner takes frequencies into account during the construction of the process model which makes the algorithm more robust towards noise in the event log. As described in Section 3.4.3, the event logs generated by *QuestionSys* do not contain noise. Additionally, infrequent behavior can be dissembled in the process model by only considering high-frequency activities. In the context of data collection scenarios, one should be careful with filtering out (infrequent) behavior because especially in the healthcare domain this infrequent behavior often contains valuable information. Nevertheless, regarding the ordering of activities when discovering the questionnaire, the Heuristic Miner returns good results. Moreover, the heuristics net can be converted into different representations such as Petri nets with very little effort, therefore allowing for a wide array of well-established techniques for generating new information.

5.1.6 The Fuzzy Miner

The Fuzzy Miner was developed by Christian W. Günther in 2007 and is the first algorithm that addresses the problems of unstructured behavior and large numbers of activities [74]. In the context of the Fuzzy Miner, a process model is interpreted as a geographic map, which allows to display, hide or cluster certain structures. Significance and correlations are used to do so. The tool *Disco*, shown in Figure 3.6, uses the Fuzzy Miner to generate a process model and allows to adjust the two metrics Activities and Paths, both with a scale from 100 to 0% [41]. The selected percentage determines the level of displayed detail with respect to either activities or paths. For example, does reducing the Activities percentage summarize activities whereas reducing the percentage of Paths reduces the edges within the fuzzy model. The output of the Fuzzy Miner is a fuzzy model which cannot be converted to other process modeling languages because the elements may be clustered. A fuzzy model consists of three different elements [75]:

1. *Primitive nodes* represent the observation of an event
2. *Cluster nodes* represent the observation of any number of events that are combined in that cluster
3. *Precedence relations* represent the edges of the fuzzy model which indicate that an event from class A *may* be followed by an observation of an event from class B

A fuzzy model allows animating the event log on top of the created process model. A screenshot of such an animation is provided on the right side of Figure 5.9. This allows for a much easier first impression of the process and also allows to get a better feeling of the dynamics within the process and its behavior. When comparing the Fuzzy Miner with the Heuristic Miner, the Fuzzy Miner is also capable of leaving out less important activities even if their frequency within the event log is high. Moreover, the Fuzzy Miner is suitable for mining less structured processes with a large amount of unstructured and conflicting behavior [76]. *Disco*, as described in Section 3.5.2, uses this discovery algorithm and its commercial success proves that the concept works. For more information about the Fuzzy Miner, please be referred to [41, 74, 75].

5.1.7 The Fuzzy Miner in the Context of Data Collection Scenarios

Figure 5.9 shows two perspectives (left and center) and an animation (right) of the Fuzzy Miner provided by the process mining tool *Disco*. Temporal information is added to the previously used event log to display the performance perspective and allow for the animation of the process. Both the slider for Activities as well as the one for Paths are set to 100 %. One of the downsides of the Fuzzy Miner is, that there is no explicit differentiation between XOR and AND constructs. The models in Figure 5.9 also display this. As shown in the left fuzzy model of Figure 5.9, both edges, and activities contain a frequency label. Activities `A,B,D` and `E` have been executed twice, while activities `Q,C` and `X` have been executed four times. The same interpretation holds for the labels on the edges. Moreover, the color of an activity and the thickness of an edge indicates their frequency. In the context of questionnaires, this provides a very good impression on the most common paths through the questionnaire.

Interpreting the performance perspective is done likewise. The darker the red color of an edge, the longer the time between the two answers given during the questionnaire. To give an example, the time elapsed between answering questions `Q` and `B` was 5.1 minutes. Please note that each activity has a duration of *instant*. The reason for this is, that the event log does only contain a single entry for each activity within each instance, similar to the results log. Additional lifecycle information like *start, in progress or completed* of each question needs to be captured in the event log to display the duration of a question. An example is provided in Section 7.1. Nevertheless, this information helps to reveal certain areas of the questionnaire in which participants need more than the expected time to answer a question. For further insights on this performance perspective please be referred to Section 7.1.

The most exciting aspect of the Fuzzy Miner is its ability to animate a process model with respect to the event log. In order to perform an animation, the event log must contain time information. If this is the case, the animation may be used to make results from the questionnaire more trustworthy in a sense that the whole data collection process can be replayed interactively. In the context of medical studies, for example, this feature makes it incredibly easy to replicate which question was answered by which patient at what point

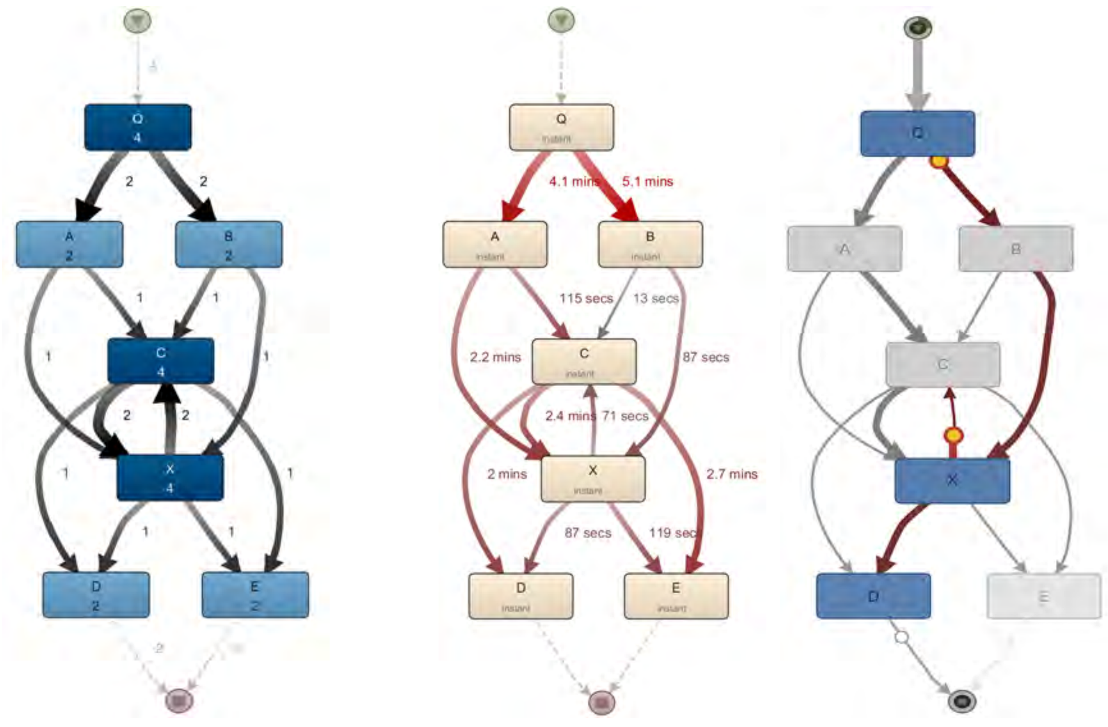


Figure 5.9: Frequency perspective (left), performance perspective (center) and animation (right) of a fuzzy model

in time. An example screenshot of an animation is provided in the right fuzzy model of Figure 5.9. In the animation of Figure 5.9, currently, three participants are answering the questionnaire, represented as circles. The first one answered question Q and decided to answer question B, the second participant answered question X and is now answering question C. The third one finished answering question D and has therefore completed the questionnaire.

5.1.8 Genetic Process Mining

After introducing two algorithms with a local strategy in the Sections 5.1.1 and 5.1.4 and a combination of local and global in Section 5.1.6, this section introduces genetic process mining as a very global approach. Compared to local strategies, global strategies try to find the process model with a one strike search. In general process mining can be seen as a search for the most appropriate model for each event log. Genetic process mining uses techniques from computational intelligence and works similar to other genetic algorithms. In *ProM*, the plug-in *Mine a Process Tree with ETMd* can be used to mine a process model with a genetic approach. The resulting process tree can then be translated into any other process modeling language such as Petri nets.

This section introduces the Genetic Miner. Its concept is based on the four steps of *initialization, selection, reproduction*, and *termination*. Then the Genetic Miner is applied in the context of *QuestionSys*.

The concept of Genetic Process Mining

By applying an algorithm with a behavior similar to the one observed in *Darwin's evolution theory* [77], genetic process mining is able to discover a process model from an event log by identifying and mutating the best process models from each generation repeatedly. Similar to other genetic algorithms the four main steps, namely *initialization, selection, reproduction*, and *termination* are executed. This section introduces the steps necessary when applying the genetic process miner as described in [14, 73]. Figure 5.10 provides a graphical representation of the four steps.

1. Initialization

   In this step, the initial population is created. In the context of process mining, this initial population consists of different process models which are created *randomly* from the activity labels available in the event log. The behavior of most generated process models may not have much in common with the event log, but some of the created individuals may have parts that conform with the event log [73].

2. Selection

   In the second step, the fitness is computed for each individual process model within a generation. This fitness represents how good each of the process models fits the event log. A simple fitness formula, for example, the proportion of traces that can be replayed, is not sufficient in this context as it is possible that no model can replay a trace, or that a very general model like the *"flower model"* returns a high fitness even though this model clearly is not correct. Therefore, a fitness function that rewards partial correctness of the process model in regard to all four quality criteria introduced in Section 4.3.1 is needed. For a more detailed explanation of the fitness metric used in this step, please be referred to [73]. Within each generation, only the best fitting process models are moved into the next generation. This is called *elitism*. Through *tournaments*, parents are selected which are then used to create new process models. The combination of *elitism* and *tournaments* mirrors the effect observed in the evolution theory and is called *survival of the fittest*. Only the most suitable process models are used to create new process models for the next generation. In Figure 5.10 these are represented as the "parents" while the others are denoted as "dead individuals".

3. Reproduction

   The third step of the Genetic Miner is reproduction. In this step, the parents identified earlier are used to create new process models through *crossover* and *mutation*. During *crossover*, two process models are used to create two new models which will then end up in the children pool in Figure 5.10. The new models are called children and contain behavior from both of the parents. These children are then *mutated* by randomly adding or removing causal dependencies. It is very important to add new behavior within the next generation because otherwise no behavior outside of the parent generation can be generated. So, leaving this step out would make it impossible to discover behavior, that may improve the process model. In other words, *crossover* recombines the fittest process models from a generation hoping that the recombination will generate an even better fitting model, and *mutation* changes a minor detail within the process model hoping to insert useful behavior into a generation [78]. After the new generation is created through *crossover*, *mutation*, and *elitism* the fitness is again computed for each process model, restarting the loop [73].

4. Termination

   During the previously described steps, termination of the algorithm was never a topic. As a result, the algorithm described so far would run infinitely long which makes it necessary to define its termination. The Genetic Miner terminates once a desired level of fitness is reached. As both the initial population as well as the mutation are random, this may take a considerable amount of time, or doesn't return a satisfying process model at all. Consequently, other termination criteria need to be added. Such criteria might be a maximum number of created generations or if the process model did not improve within the previous X generations. Once the algorithm terminates, the currently best fitting model is returned [14].

Figure 5.10 represents the described approach.

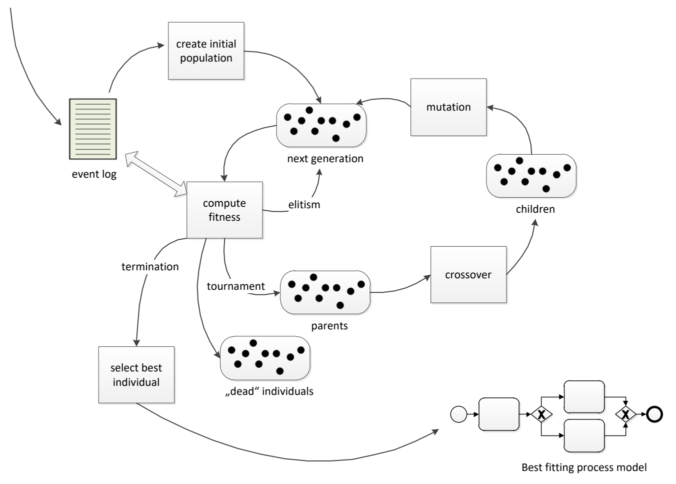


Figure 5.10: Overview of the approach used for genetic process mining [14]

Based on an event log the initial population is created randomly, and the fitness is computed for each individual. Through *elitism*, the best process models are included in the next generation. With the help of tournaments, parents are selected to produce children through *crossover*. By mutation of these children, new behavior is introduced into the next generation. Non-fitting process models are ignored, represented as the dead individuals. The same procedure is then applied to the next generation until the algorithm terminates and the best individual is returned.

For additional information about genetic process mining please be referred to [14, 73, 78, 79, 80, 81].

5.1.9 Genetic Process Mining in the Context of Data Collection Scenarios

The Genetic Process Miner is implemented in *ProM 6* as the "Mine a Process Tree with ETMd" plug-in. ETM stands for Evolutionary Tree Miner. Due to bad performance, the initial Genetic Miner plug-in was removed from *ProM 6*. An older version of *ProM* has to be used if this plug-in is required. The results presented in this section are generated with the *"Mine a Process Tree with ETMd"* plug-in under *ProM 6*. As the name suggests, the Evolutionary Tree Miner is a genetic algorithm which discovers process trees. *ProM* is used to translate the generated process trees into Petri nets.

Figures 5.11 - 5.16 show different results of the Genetic Miner, with respect to the number of generated generations. Each generation consists of 20 process models. To avoid quality degeneration, the 5 best fitting candidates are included in the next generation though *elitism*. The same event log as in the previous sections is used as input.

Figure 5.11 shows a discovered questionnaire model after only one generation. Therefore, it is simply the best fitting model within the initial population and should not be used. Its main purpose is to illustrate the first step of the algorithm.

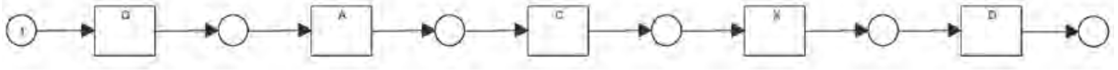


Figure 5.11: Applying the Evolutionary Tree Miner with 1 generation

Figure 5.12 displays a discovered questionnaire after ten generations. In comparison to Figure 5.11, this questionnaire already contains much more behavior such as the XOR-construct between questions `A` and `B` as well as `D` and `E`. Moreover, the AND-Split between questions `X` and `C` is also discovered correctly, but question `X` is duplicated.

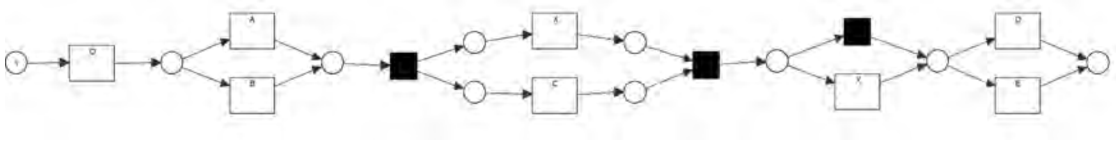


Figure 5.12: Applying the Evolutionary Tree Miner with 10 generations

Figure 5.13 shows a discovered questionnaire after 100 generations. Here the XOR-construct between questions `A` and `B` as well as the AND-construct between `C` and `X` are correctly discovered and there are no duplicate questions. The XOR-construct between questions `E` and `D` is not discovered, as it is represented in sequence.

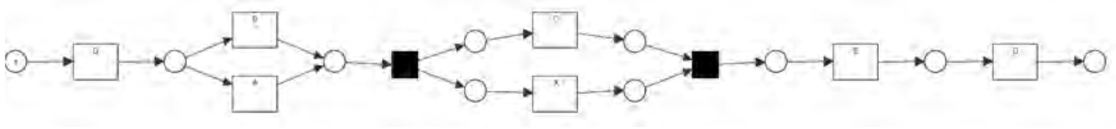


Figure 5.13: Applying the Evolutionary Tree Miner with 100 generations

Figures 5.14 and 5.15 represent two discovered questionnaires after 1.000 generations. Whereas the questionnaire discovered in Figure 5.14 represents a "good" result in a sense that all splits and joins have been identified correctly, the same algorithm with the same settings discovered a different questionnaire in Figure 5.15. This is due to the high amount of randomness within both initial populations as well as the mutation phases.

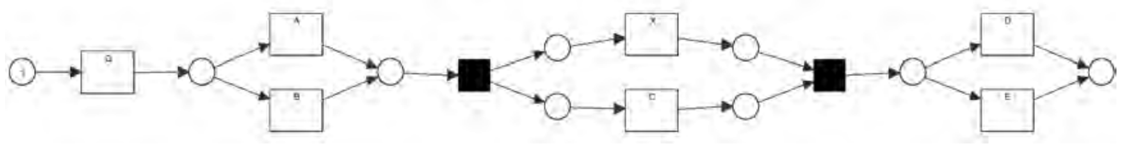


Figure 5.14: Applying the Evolutionary Tree Miner with 1.000 generations (1)

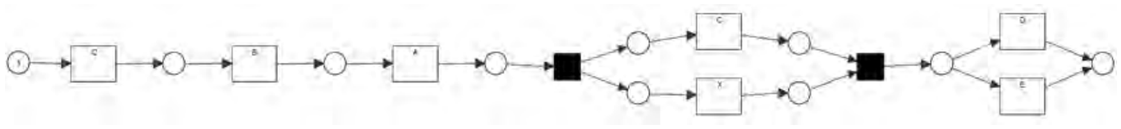


Figure 5.15: Applying the Evolutionary Tree Miner with 1.000 generations (2)

The result displayed in Figure 5.16 shows a discovered questionnaire after 10.000 generations. It is fairly similar to the result generated from 100 generations, with just a different ordering of questions D and E.

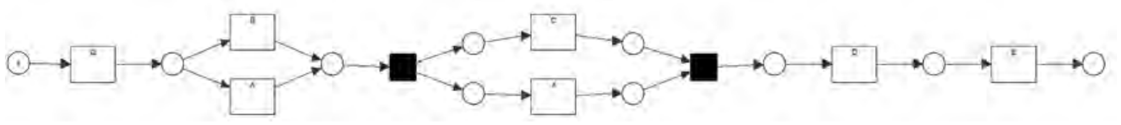


Figure 5.16: Applying the Evolutionary Tree Miner with 10.000 generations

The results displayed in Figures 5.11 - 5.16, show that the Genetic Mining can successfully discover process models, even in the context of questionnaires. This approach may be especially attractive if the event log contains noise or is incomplete. There are some arguments against using a Genetic Miner in the context of *QuestionSys*. First of all, the event logs generated in the context of *QuestionSys* are complete and noise free. Moreover, Figures 5.11 - 5.16 show that even with a high amount of generations, the genetic algorithm cannot guarantee that a good questionnaire model is found. Another aspect that should not be neglected is the time needed to discover the process model. Whereas algorithms based on ordering relations such as the Alpha or Heuristic Miner are fast in constructing a model, the genetic algorithm may take a long time, and still does not guarantee a better result than the other algorithms. Furthermore, questionnaires often include constraints regarding the ordering of certain questions due to the questions belonging to the same category such as alcohol, drug consumption or pre-existing conditions. With the presented algorithm, it is impossible to take such constraints into account when discovering a questionnaire.

5.2 Comparison of different Process Discovery Algorithms

After introducing various process discovery algorithms in this chapter, this section aims to provide a brief overview of these discovery algorithms. Table 5.2 summarizes the different algorithms introduced in this chapter.

Within an academic context, most people read about the Alpha Algorithm and keep using it, as it is easy to understand and interesting properties can be proven around it [57]. These different properties make the Alpha Algorithm *"beautiful"* from an academic point of view. Also, in the context of *QuestionSys*, the event logs are complete and noise free, which is why the Alpha Algorithm, or one of its extensions, may be applied successfully. When dealing with real-life logs, the Alpha Algorithm is almost never the right choice, because the event logs will almost certainly contain both incompleteness and noise, and as a result, the generated process models won't be good.

As the second process mining algorithm, the Heuristic Miner solves many problems of the basic Alpha Algorithm. It derives the XOR and AND connectors from dependency relations and is able to abstract from infrequent behavior by leaving out edges with low frequencies. Moreover, it can deal with noise, which makes it suitable for many real-life event logs. Additionally, the heuristics net can be converted into other types of process models, like Petri nets, allowing for further analysis with tools like *ProM*.

The Fuzzy Miner was specially designed to deal with highly unstructured behavior and large numbers of activities. By using significance and correlation metrics the Fuzzy Miner is able to interactively simplify a process model to a desired level of abstraction. This allows to leave out less important activities even if they have a high frequency. Moreover, a fuzzy model allows to animate an event log on a process model, but unfortunately, cannot be converted to other process modeling languages.

The Genetic Miner solves many problems, but is too complex to set up in order to be used in real life situations. Additionally, due to the high amount of randomness, it cannot guarantee good results and reproducibility.

	Basic Alpha Algorithm	**Heuristic Miner**	**Fuzzy Miner**	**Genetic Miner**
Description	First mining approach which allows to discover a WorkFlow net from an event log	Generates a process model based on different frequency metrics	Controlling significance cutoff allows to zoom and show different importance levels	Discovers all common control-flow structures and provides a view of frequency for successions and tasks
Strategy	Local	Local	Local and global	Global
Output	WorkFlow net	Heuristic/Causal Net	Fuzzy model	Process tree
Challenges	Noise Incompleteness Length one loops Length two loops Non-free choice	Split and Join rules are only considered locally which results in nets that are not sound	Fuzzy model cannot be converted No differentiation between XOR & AND	Might not terminate, take a lot of time, may not return a good model
Result	Create a petri net from an event log Extensions are able to tackle some of the challenges	Can successfully mine long distance dependencies, but sometimes generates too many dependencies	Can handle noise and incompleteness in the log, and mines most dependencies correctly	Results may vary May also discover behavior not in the original model
Event logs	Cannot handle incompleteness or noise	Can to a certain degree deal with incompleteness	Can handle incompleteness	Can deal with noise and incompleteness

Table 5.2: Summary of Process Discovery algorithms [76]

6 Conformance Checking

After covering the topic of control-flow discovery in the previous chapter, this chapter focuses on another situation. Both the process model and the event log are available and it is irrelevant if the process model was constructed by hand or discovered. This situation translates to the second form of process mining which is conformance checking. Conformance checking allows to find discrepancies and commonalities between the process model and the event log. The event log represents the actual behavior while the process model represents the modeled behavior. One of the more traditional use cases of conformance checking is fraud or inefficiency detection during replay. Moreover, conformance checking can be used to measure the performance of a mining algorithm by comparing the discovered model with the actual event log, or to repair a process model to be more in line with reality.

To be able to apply conformance checking algorithms, it is necessary to relate the events within an event log to the activities of the corresponding process model. There are three different ways of doing so, which are shown in Figure 6.1 [14].

The first way is called play-in. During play-in, information from the event log is used to generate the process model. Process discovery algorithms may be categorized in here.

The second way is not only the opposite of play-in but also the traditional use of process models. In this case, example behavior from the process model is generated by playing out the process model. In doing so, example behavior of the process model is generated and documented in an event log. This is called play-out.

The third way is replay. Replay utilizes both the process model and the event log to check how good they fit together. Moreover, replay can also be used to identify

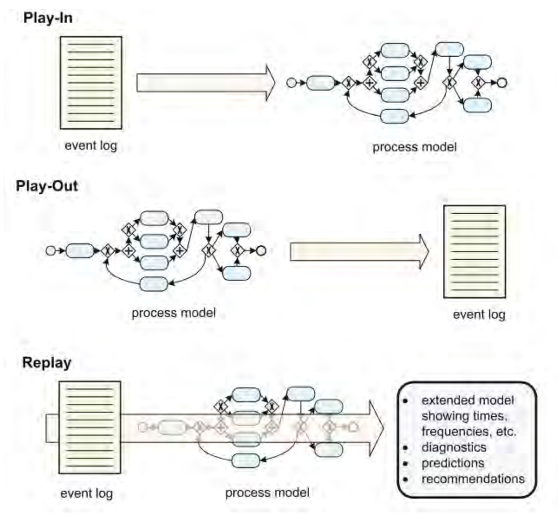


Figure 6.1: Three ways of relating event-logs with a model [14]

bottlenecks, support decision analysis, predict the behavior of running process instances or recommend suitable actions. During this chapter, the focus will be on replay as a technique to check conformance.

Figure 6.2 shows the general idea of conformance checking. By comparing the actual behavior documented in an event log with the possible behavior of a process model, discrepancies, as well as commonalities, can be identified. These results may be differentiated in global and local conformance measures. Global measures lead to more general statements like "42 % of the cases in the event log fit into the model", whereas local diagnostics drill down deeper into the model and analyze specific activities within the process model. A resulting statement from a local diagnostic could be "Question number 12 was answered 3 times although this was not allowed at this point in the process model" [14].

Different points of view need to be taken into account when interpreting deviations found during conformance checking. It is always possible that the process model is wrong and doesn't reflect the reality, or that the event log deviates from the process model because corrective measures are needed. Since both perspectives are important, and conformance checking should support both aspects, Figure 6.2 indicates deviations in both the event log as well as the process model [14].

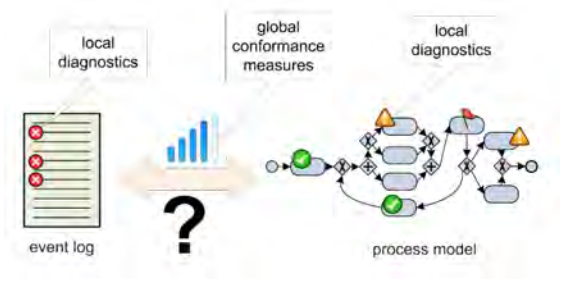


Figure 6.2: Conformance checking: global conformance and local diagnostics [14]

This chapter is structured as follows. In general, there are three approaches to conformance checking, which will be described. The first approach abstracts the behavior seen in the event log and the process model to then compare it. The notion of footprints can be seen as such an approach and is described in Section 6.1.

A second approach is replaying the event log on the process model in order to find deviations. The Token Replay Algorithm uses this approach and is described in Section 6.2.

The third approach is the most advanced approach of the three. An optimal alignment between the event log and the most similar behavior represented by the process model is computed to detect deviations. This approach is described in Section 6.3 [82].

Additionally, an approach based on Linear Temporal logic is presented in Section 6.4.

6.1 Footprint Comparison

A footprint is a matrix displaying causal dependencies, introduced in Section 5.1.1, and contains the relations $a \rightarrow b$, $a\#b$ and $a||b$. It is independent of the process modeling language because it relates to the behavior instead of the language [83, 84].

Footprints can be derived from an event log as well as a process model. The footprint of an event log may be derived from the "directly follows" relation ($a > b$) previously introduced in Section 5.1.1. The footprint of a process model may be generated by playing out the process model, recording an event log in a way that each possible path through the model is taken at least once. Based on this event log, a footprint can be derived which represents the behavior of the process model.

When comparing the two matrices, differences are revealed.

On the one hand, may this information be used to get detailed diagnostics on which parts of the log or model are not conforming, on the other hand, a fitness measure is easily computed by dividing the found differences by the total amount of cells in the matrix. For example, if 12 out of 64 entries differ, the fitness is $\frac{12}{64} = 0.8125$ or 81,25 %.

The possibility that footprints can be generated from both an event log and a process model yield interesting potential. On the one hand are the relations between events documented in the event log described in the footprint of an event log, on the other hand does the footprint of the process model represent how activities in the model relate to each other [84].

This allows to compare all three combinations of event log and process model. Comparing the footprints of a process model and an event log allows checking whether they represent the same ordering of activities, and is already explained above.

Comparing two process models allows to estimate their similarity. Even if their graphical representation is different, they may still be (partly) equivalent regarding their behavior. Footprints may also be used when changing a process model to better represent the observed behavior represented in the event log, and the amount of change needs to be quantified, as the changes will be represented within the footprints. By replaying all possible behavior from the old and the new process model, footprints are derived from both process models. A comparison of these two footprints identifies the changes.

Comparing two event logs helps to identify so-called concept drifts. A concept drift refers to a situation in which the observed process changes during analysis [14].

Of course, the footprint representation is only one of many representations of an event log, and other representations may suit the notion of conformance better. In theory, any temporal or heuristic measure can be used instead of the "directly follows" relation or the logic may also be extended with a time window. It is possible to extend the distance between activities when deriving the causal relations. A possible extension could be:

"activity a was followed by activity b within 3 steps and the other way around in more than 50% of the cases". In a similar fashion as with the Heuristic Miner, frequencies might be taken into account when constructing the footprint matrix [14, 85, 86].

6.1.1 Footprint Comparison in the Context of Data Collection Scenarios

In the context of data collection scenarios, the comparison between two models can be used to compare two different versions of a questionnaire. This enables quantifying the behavioral change between different iterations of a questionnaire. Being able to quantify both the fitness and the change between two iterations of the questionnaire allows measuring the effectiveness of changes made between multiple versions of a questionnaire. Comparing two event logs allows to identify the differences between different iterations of a questionnaire, e.g. the first run of the questionnaire, second run, and so on. As the footprint may also be derived on the instance level or of different groups within an event log, e.g. from a special user or a department within a company that answered the questionnaire, its behavioral change may also be derived over time. The simple comparison between the log and the process model as described in Section 6.1 might help to get a first impression of the fitness. More robust approaches to quantify fitness are presented in the next sections. Additionally, questionnaires are often times specially tailored towards a topic, so extending the representation may lead to new relationships between answers. Extending the distance of the observed relations may also lead to new insights within a questionnaire.

6.2 The Token Replay Algorithm

As described in Section 4.3.1, there exist four quality criteria. In this section the notion of fitness is introduced, which represents the proportion of the behavior seen in the log that can also be generated by the process model.

A very a naive approach for checking conformance is to only count the traces in the log file which can be replayed correctly on the model. If 97 out of 100 traces can be replayed correctly, this would then lead to a fitness of 97%. When a deviation between model and log occurs, the corresponding instance is not further considered. This allows a rough first impression, which, however, has some weaknesses. It cannot be differentiated *how bad* the trace fits the log. There should be some kind of gradation regarding the severity of the deviation. The fitness between a trace and the model should be higher if the deviation is smaller. While the naive approach neglects all traces that do not fit perfectly into the model, this approach clearly does not take any severity into account, and therefore, cannot differentiate between different deviations.

The Token Replay Algorithm can distinguish between different deviations and can, therefore, take severity into account, by quantifying the mismatch.

Because we have already introduced another notion of fitness earlier in Section 5.1.8, please note that this notion of fitness is not sufficient enough for the selection phase of Genetic Mining. For other aspects of process mining, the Token Replay Algorithm is a sufficient fitness measure.

The idea behind the Token Replay Algorithm is really simple [87, 88]. Four different counters are recorded for each activity in the log.

`p` = Produced tokens (incremented if a new token is generated)

`c` = Consumed tokens (incremented if a token is consumed)

`m` = Missing tokens (incremented if the log indicates firing a transition while there are not enough tokens in preceding places)

`r` = Remaining tokens (tokens that are remaining in the model after the trace has been replayed)

Initially p and c are set to 0. At the start, the environment produces one token in the start place and as a result p is increased by 1. Then, for each of the activities in the log the respective activity in the net is executed. This means that all tokens from preceding places are consumed, and new tokens are added to the succeeding places. Each consumed token increments the consumed counter c, and each produced token increments the produced counter p.

If the event log indicates that an activity is executed while the respective transition in the net is missing some tokens to be enabled, it is still executed as if the token would be in the correct place. For each of the missing tokens the missing counter m is then incremented by one.

After all activities from a trace are replayed as described, all tokens that are still remaining in the process model are counted and the remaining counter r is incremented accordingly. As a last step, the environment needs to consume a token form the final place, which increments c by 1.

With the gathered information from the four counters, the fitness of a specific trace with the model can be calculated. The fitness of a trace σ and a workflow net N is defined as follows:

$$fitness(\sigma, N) = \frac{1}{2} * (1 - \frac{m}{c}) + \frac{1}{2} * (1 - \frac{r}{p})$$

The first part of this formula computes the relation between missing and consumed tokens. If there are no missing tokens, this part equals 1, and decreases with more missing tokens. The last part is defined equally in regard to remaining tokens. This formula values missing tokens equal to remaining tokens, which both represent unwanted behavior and therefore reduces the fitness of model and trace [14, 87].

In a similar way the fitness of a log file containing various traces can be calculated. Therefore, every trace within the log needs to be replayed as described earlier. Then the fitness between a log L and a workflow net N can be computed with the corresponding formula:

$$fitness(L, N) = \frac{1}{2} * (1 - \frac{\sum\limits_{\sigma \in L} L(\sigma) * m_{N,\sigma}}{\sum\limits_{\sigma \in L} L(\sigma) * c_{N,\sigma}}) + \frac{1}{2} * (1 - \frac{\sum\limits_{\sigma \in L} L(\sigma) * r_{N,\sigma}}{\sum\limits_{\sigma \in L} L(\sigma) * p_{N,\sigma}})$$

This formula has the same properties as described before. The value of $fitness(L, N)$ is always between 1 (perfect fitness; no missing or remaining tokens in any trace) and 0 (bad fitness - all produced tokens are remaining, all consumed tokens are missing), and decreases the more missing and remaining token occur.

Another aspect of the Token Replay Algorithm is that it also allows for diagnostics. It is rather simple to assign the amount of tokens that passed each edge of the Petri net. They represent the amount of produced and consumed tokens, and therefore indicate how frequent a specific path was taken within the represented log. Places within the Petri net can also be tagged with the information generated by the missing and remaining tokens. Aggregating this information displays where which part of the model does not conform with the log or which parts of the log deviate from the model. Additionally, this allows to quantify the severity of the deviation. On the one hand, if the amount of missing or remaining tokens is low, the severity of the deviation is low. On the other hand, if the amount of missing or remaining tokens is high, the severity of the deviation is high and can be an indication that changes in the model are required or there are problems with the log [14, 89, 90].

6.2.1 The Token Replay Algorithm in the Context of Data Collection Scenarios

As the process engine of *QuestionSys* does not allow for any deviations from the process model, one could argue that the fitness of the event log and the process model will always be 100 %. This is correct as long as only completed instances are taken into account. If aborted instances are also taken into account, the fitness may be below 100 % due to the fact that an aborted instance result in remaining tokens in the net and additionally results in a missing token in the sink place (the token that has to be consumed by the environment in the last step).

This information can be used for different metrics. Statistics like, for example a completion rate (e.g. $1 - (\frac{\mathtt{m}_{sink}}{\#_{instances}})$ allow to estimate the percentage of participants answered the questionnaire. Furthermore, the information gained by the remaining tokens can be used to evaluate where most of the participants decided to stop answering the questionnaire.

If it becomes evident that at a certain point within the questionnaire a lot of participants decide to drop out, it might be necessary to improve the structure of the questionnaire. Some participants might be offended a question and as a result decide to stop answering the questionnaire, others may feel like the amount of questions is too high and, as a result, quit. A structural change might be as simple as changing the order of the questions, or the wording of a question. Having a lower rate of dropouts within a questionnaire is obviously better because it allows to collect more complete data sets from the participants. This makes the whole data collection process way more efficient and reduces the time needed for data cleaning preparing.

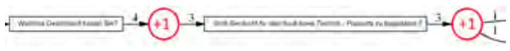


Figure 6.3: Two places containing a remaining token

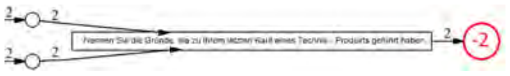


Figure 6.4: One place missing two tokens

Figure 6.4 and Figure 6.3 represent an example result generated with *ProM*. Please note that for reasons of simplicity only the parts of the net in which deviations were identified are shown. The fitness between the event log and the model is 93.55%. Additionally, we can see that there is a remaining token in each of the places between question number three and four and between question number four and the respective choice construct, displayed in Figure 6.3. Both of these tokens are also denoted as missing in the last place of the net, displayed in Figure 6.4. The questionnaire model has been discovered by the Alpha Algorithm, and two cases have been added to the event log representing participants dropping out of the questionnaire in a way that they only answer a part of the questions from the questionnaire. These two cases represent the behavior described above in which a participant dropped out of the questionnaire.

6.3 Alignments

The algorithm described in Section 6.2 will work reliably in the context of *QuestionSys*, because the event log will always be complete, and the engine does not allow for any deviations [2, 37]. In reality, this is not always the case which is why the Token Replay Algorithm has some drawbacks. One of the drawbacks is, that it is specific to Petri nets, so if a mining algorithm that does not produce a Petri net is used during process discovery, or the existing process model is not modeled as a Petri net, it is not possible to use the Token Replay Algorithm without converting the model beforehand. Additionally, if the process model and the event log are deviating heavily, the process model is flooded with tokens. Having many tokens in a Petri net allows for any behavior possible in the model and therefore no differentiation between *correct* and *incorrect* behavior is possible.

Conformance checking with the Token Replay Algorithm is no longer supported in newer versions of *ProM* because more sophisticated and robust algorithms such as Alignments have been developed.

To create an alignment between the process model and the event log, it is necessary to relate actions within the event log to the activities represented by the process model. This sounds like an easy task but is a very difficult task once the event log and model start to deviate [82].

When dealing with Alignments, the three different moves *Synchronous move, Move on model only* and *Move on log only* need to be taken into account. They represent the different behavior that may occur and has been introduced in [91].

1. *Synchronous move*

   A synchronous move represents that a move within both the process model and the event log was taken. In other words, they fit together.

2. *Move on model only (Model moves)*

   Moves on model only represent that an activity in the process model had to be executed without it being documented in the event log.

3. *Move on log only (Log moves)*

   Moves on log only may be seen as the opposite of modes on model only. An activity within the event log is executed but no activity in the process model respectively.

Model moves and log moves represent deviations and should, therefore, reduce the fitness. By calculating costs for log moves and model moves, deviations can be represented by a lower fitness. A possible fitness function is defined in [91]. Of course, different costs can be assumed for each activity and each type of move. Activity `X` may have a cost of 5 when there is a model move, while activity `Z` has a cost of 1 if a log move occurs. This allows for an even more precise representation and more important a weighting of deviations, not only on the type level but also on the activity level.

Alignments allow for more detailed diagnostics based on the instance level, which may be aggregated into diagnostics regarding the whole process. In addition, Alignments can indicate which activities are often skipped (represented by model moves) or that an activity is frequently executed at times where it is not supposed to, according to the process model (represented by log moves). This allows to relate the behavior from the event log to the process model in a more precise way [14].

Calculating Alignments is not a trivial task, because there may be multiple Alignments for a single instance, and the goal is to find the best fitting one. Some Alignments are not optimal in a sense that there is an alignment with lower cost, especially if the costs are customized. The implementation in *ProM* guarantees to return an optimal alignment [91]. If you are interested in the formal definition of Alignments please be referred to [91, 92].

6.3.1 Alignments in the Context of Data Collection Scenarios

When using a version of *ProM* that is newer than version 5.2, the conformance checking tool automatically uses Alignments instead of the Token Replay Algorithm. With the same event log and process model as already used in Section 6.2, the result looks a bit different this time. Figure 6.5 displays the result when checking conformance using Alignments instead of the Token Replay Algorithm.

The result presented in Figure 6.5 contains more than one information. It incorporates the frequency of different questions by changing the color of the question in the questionnaire model. A darker color of a question represents a higher frequency. Some of the questions in Figure 6.5 have a red border, indicating a model move. The green and purple bar on the bottom represents the ratio between model moves and synchronous moves. Based on this result, there was a model move in the fourth question, resulting from one participant dropping out of the questionnaire. Each subsequent question indicates a model move because Alignments do not recognize the abortion of a process instance. Instead, the instance is completed via model moves, reducing the fitness. Looking at the posterior questions of the questionnaire, they indicate that there are 50% (2 out of 4) model moves, resulting from the two dropouts added to the event log. Of course, it is possible to also calculate the fitness, and in this scenario, the fitness is significantly lower than compared to the Token Replay Algorithm. While the fitness using the Token Replay Algorithm was 93.55%, with Alignments the fitness is only 77.88% [92].

Figure 6.5 is created with the *ProM* plug-in called *Replay a Log on Petri Nets for Conformance Analysis*.

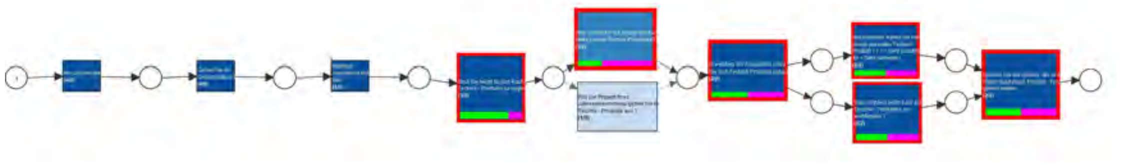


Figure 6.5: Conformance checking using Alignments

6.4 The Linear Temporal Logic Checker

While the previously mentioned algorithms, Footprints comparison (Section 6.1), Token Replay (Section 6.2) and Alignments (Section 6.3), compare the event log with the process model as a whole to estimate how well they fit together, the Linear Temporal Logic approach pursues another goal. In many cases, there are other constraints towards the process, like the "4-eyes principle" in which certain activities are not allowed to be executed by the same person. Constraints like this can be verified directly on the event log, without even taking the process model into account. As a result, this approach doesn't fit 100% into conformance checking because conformance checking relates the behavior within the log and the model with one another. Nevertheless, this approach can yield high potential in the context of process mining as it allows to control for undesired behavior or check constraints regarding the whole process (in the context of data collection scenarios the whole questionnaire). How the LTL-based approach can be used in the context of process mining, is described in detail in [93].

The idea of Linear Temporal Logic (LTL) is to extend the classical logical operators by temporal aspects. These temporal operators may be always ($\Box$), eventually ($\diamond$), until ($\sqcup$), weak until (W) or next time ($\bigcirc$) [94].

Linear Temporal Logic is not limited to checking if the execution of activities fits with the expected behavior. It allows to take other attributes like the originator or data into account. This way other aspects, like temporal constraints (e.g. the time between executing two activities should not be longer than 15 minutes) or data constraints (e.g. if the patient is female and has stomach ache a pregnancy test has to be executed) can be defined and checked for conformance with these rules directly on the event log. The LTL checker will then separate the initial event log into two event logs. One of the event logs contains all compliant traces while the other one contains the non-compliant ones. Additionally, there is a lot of well-established research in the area of logical expression which can also be applied in this context [14, 93, 95].

Currently, there are 52 common properties already implemented in *ProM* that may be used to check certain constraints. These properties can be used without any prior knowledge about the LTL language. It is also possible to create new, and situation specific, formulas for the LTL-Checker in *ProM* [93].

6.4.1 The LTL-Checker in the Context of Data Collection Scenarios

The most intuitive approach would be to use some of the predefined LTL formulas in *ProM*. Providing a general example is hard because these constraints are highly dependent on the questionnaire and its structure.

When conducting a study which uses a questionnaire to collect data, there may be assumptions which should be tested. For this case, the LTL checker offers a great and intuitive approach on checking these assumptions without even having to apply a process discovery algorithm. If the questionnaire is about employee satisfaction, and the assumption is that free beverages and fruits help to increase employee satisfaction it can easily be checked which share of instances in the event log satisfy the corresponding LTL-formula. The LTL checker then separates the event log into two event logs which may then be used as an indication which share of instances complies with the behavior from the assumption. This way it is easy to see if the assumption could be valid or not. Each of the separated event logs may again be used for further analysis. The formulas of the LTL checker are not restricted to questions and their ordering, but may also be used to analyze the answers given, the time needed to answer different questions or how often the answer has been changed during execution of the questionnaire as long as the corresponding information is documented in the event log. It may be interesting to further investigate the instances in which a certain question was answered three or more times. Moreover, the originator may also be used within the formula as long as the event log contains such information.

Another use case of the LTL checker is to filter out dropouts. One of the predefined formulas allows to check for the last activity within the event log. In the context of a questionnaire, this may be used to either filter an event log for desired behavior or remove certain behavior such as dropouts to improve the questionnaire discovered by a process discovery algorithm.

Figure 6.6 shows a screenshot of *ProM*. First, an event log is imported as a CSV-file, which is then converted into a XES event log. Based on this XES event log, the LTL Checker was applied to separate the event log into compliant and non-compliant traces. Each of the different event logs may then again be used for further analysis.

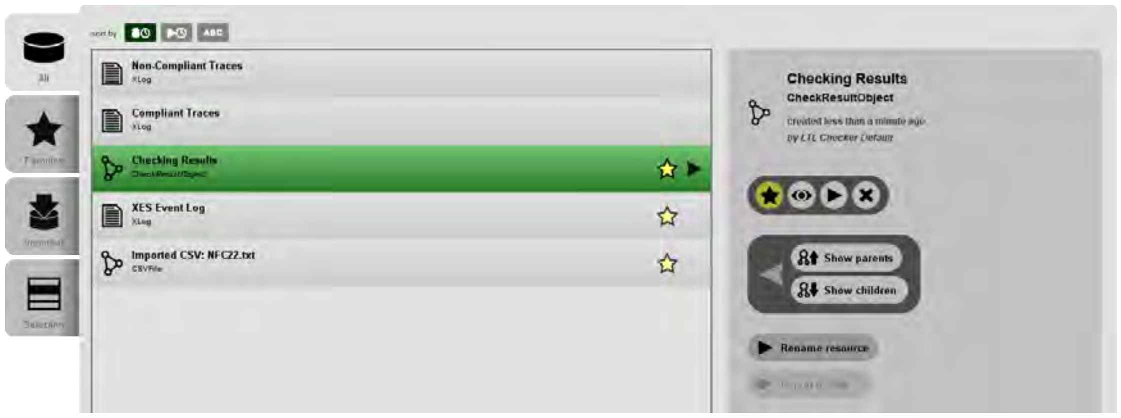


Figure 6.6: Original CSV file, the resulting XES event log and the two results from applying the LTL checker in *ProM*

7

Enhancement

Within Chapter 5 *Process Discovery* was introduced, which can be used to generate process models from information contained in an event log by applying various algorithms. Different algorithms have been introduced which are more or less useful in certain scenarios.

In the previous chapter, *Conformance Checking* was introduced, which is used to relate an event log and a process model indicating improvement potential. Moreover, Conformance Checking algorithms may be used to evaluate the quality of existing or discovered process models.

Both *Process Discovery* and *Conformance Checking* only use a portion of the information contained in an event log. In many cases, an activity within an event log can contain additional information like an originator, data attributes, sensor data, timestamps or even geographic information if the data is collected on mobile devices. While this information is often not considered during both *Process Discovery* and *Conformance Checking*, this chapter introduces enhancement techniques which can be used to enrich a process model with such additional information. This chapter illustrates why process mining is often referred to as the *bridge between Data and Process Science* [14].

Similar to *Conformance Checking*, Enhancement also uses both an event log and a process model as an input. Moreover, this chapter assumes that a control-flow model exists and enhancements of the process model with respect to the information contained in the event log are introduced.

This chapter is structured as follows: First, temporal information from the event log is used to enrich the process model with a performance perspective. Afterward, the concept of organizational mining is introduced, which utilizes information on who executed an activity to discover relationships between different involved persons. Last, decision trees are used to better understand which data generated during the process influences decisions made within the process instance. All three approaches may be used to enrich a process model with additional perspectives, therefore increasing its expressiveness and allows to possibly discover novel correlations within a questionnaire.

7.1 Temporal Aspects

While process discovery mainly focuses on the ordering of activities, also known as the control-flow perspective, this section will take the time perspective into account. The focus of the time perspective is on all time-related aspects of a process. Activities may occur at a certain time, take a certain amount of time until they are completed or waiting times between activities are specified. If the event log contains temporal information, like the duration of an activity or timestamps, this information can be used to enhance the process model. While some event logs may only contain date information, e.g. "14-08-2018", others provide timestamps with millisecond precision. As a result, the different granularity of timestamps needs to be taken into account during analysis. *QuestionSys* provides timestamps with millisecond precision, and this section will also assume that the event log contains such precise timestamps. To enhance a process model with temporal information, only a small modification to the Token Replay Algorithm, introduced in Section 6.2, is necessary [14].

Depending on how the event log is structured, different temporal metrics can be measured. If the event log contains lifecycle information, for example *started* and *completed* for each activity, the duration of each activity can be measured by subtracting the started timestamp from the completed timestamp [14]. Additionally, the time between two activities can be measured by comparing the completion of the first activity with the start of the next one. This then corresponds to a *waiting time* between two activities.

Enhancing a process model with information from the time perspective can be used to discover bottlenecks, or estimate the remaining time until a specific process instance is completed.

In *ProM*, there are multiple plug-ins, such as *"Replay a Log on Petri Net for Performance/Conformance Analysis"* [91] or the *"Multi-perspective Process Explorer"* [30] that are able to take temporal aspects into account.

7.1.1 Enhancing Questionnaires with Temporal Information

In the context of *QuestionSys*, the event log used to create Figure 7.1 may be derived from the results log, as this one does only contain a complete event of each question. As a result, the duration of a question cannot be calculated in a similar way as explained in Section 7.1. Because the following question is displayed immediately after the previous one is answered, the time between two questions is assumed to be 0. Hence, the time between the completion of two questions can be interpreted as the time needed to answer the second question. To compute the duration of the first question, a timestamp indicating the initial start of the whole questionnaire is necessary. An example of this is provided in Figure 7.1.

Enhancing a questionnaire with temporal information allows for many new insights in both the participant and the questionnaire itself. Figure 7.1 displays the result of applying the *"Multi-perspective Process Explorer"-Plugin* in *ProM* to an event log which has been extended with timestamps [30].

The process model displayed in Figure 7.1 is generated by the *Multi-perspective Process Explorer* plug-in of *ProM* and is enhanced in multiple ways. Each edge from a place to a transition contains two different measures. The first measure is the share of traces that pass the edge during execution, represented by a percentage. If there is no percentage assigned to an edge, the previous percentage is still correct. Additionally, the frequency is also represented by the thickness of the edge. he second measure, the one that this section is about, is a temporal one. Each edge between a place and a question is labeled with temporal information. This time represents the average time needed from answering the previous question to answering the next question. Other metrics such as the minimum, maximum, median time may also be displayed. In this example, it does on average take 4,2 minutes to answer the question about the date of birth. Afterward, it takes 1,9 minutes to answer the question about the gender.

The color of the transition represents how often the question was answered. As we are only focusing on completed instances (how to identify dropouts has been explained in Sections 6.2.1 and 6.4), all four instances of the questionnaire are completed. Furthermore, the model indicates that 50 % of the traces took the top path while the other 50% took the bottom path at the decision point. This is also shown by the color of the two involved questions as well as the color of the edges.

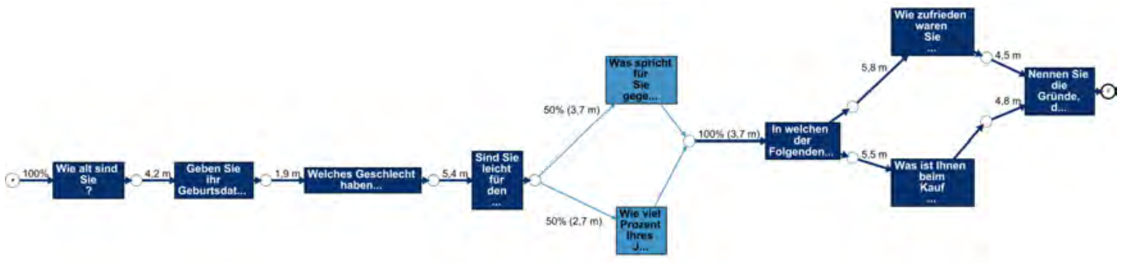

100%
Wie alt sind Sie ?
4,2 m
Geben Sie ihr Geburtsdat...
1,9 m
Welches Geschlecht haben...
5,4 m
Sind Sie leicht für den ...
50% (3,7 m)
Was spricht für Sie gege...
50% (2,7 m)
Wie viel Prozent Ihres J...
100% (3,7 m)
In welchen der Folgenden...
5,8 m
Wie zufrieden waren Sie ...
5,5 m
Was ist Ihnen beim Kauf ...
4,5 m
4,8 m
Nennen Sie die Gründe, d...

Figure 7.1: Process Model enriched with Temporal Information (average duration)

By enhancing the process model with the *time perspective*, valuable information, such as the time needed to answer each question, may be generated about the participants. It may be very difficult to collect such data from paper-based questionnaires.

Interpreting this may therefore lead to new insights for both the questionnaire as well as the participant. On the one hand, if answering a question takes only a very short time the participant might have just given a random answer. On the other hand, if answering a question takes a very long time, the question might be too difficult, too private or maybe the user interface was not optimal. Giving concrete time spans for when an answer is given *fast* or *slow* is hard as it depends heavily on many different factors such as the subject of the questionnaire, the surrounding in which the questions are answered and of course the participant himself. Enhancing the process model with the average time allows to get a good impression about the whole group of participants and may be a good starting point to drill down on those instances that took significantly more or less time.

Additionally, the average time to complete a questionnaire may also be interesting to investigate. By simply computing the duration between the first question and the last question for each participant, the average time needed can be calculated easily, and other aspects of *QuestionSys* can use this information. The average time needed to answer the questionnaire can be displayed before the start of the questionnaire, which allows participants to decide whether they have enough time to complete the questionnaire or not, therefore reducing dropouts.

Figure 7.2 shows the performance perspective of *Disco*. The perspective is similar to the performance perspective presented earlier in Chapter 5, but the event log used to discover the questionnaire is the same as in Figure 7.1. The event log has been extended with lifecycle information for each question. As a result, Figure 7.2 also indicates the average time needed to answer each question. It is possible to calculate these times based on different lifecycle transitions such as start and complete for each question in the event log. Different colors indicate how long participants need to answer a specific question. The example provided in Figure 7.2 indicates that participants need more time to answer long questions compared to questions that are formulated short and concise. The exception is the second question, which could indicate that the usability of some of the elements used to answer this question might not be ideal. Note that this is just an interpretation of this specific result, and for different questionnaires, the interpretation may differ.

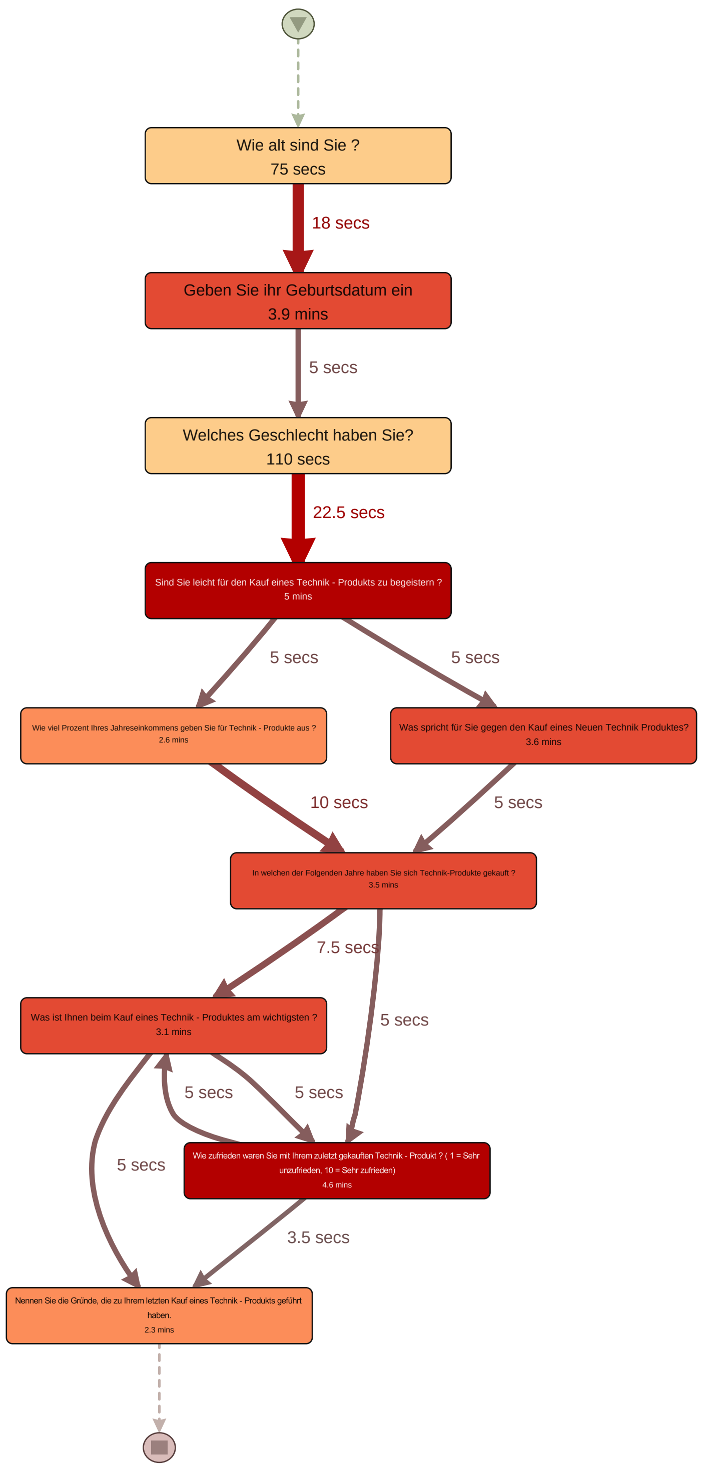

Wie alt sind Sie ?
75 secs
18 secs
Geben Sie ihr Geburtsdatum ein
3.9 mins
5 secs
Welches Geschlecht haben Sie?
110 secs
22.5 secs
Sind Sie leicht für den Kauf eines Technik - Produkts zu begeistern ?
5 mins
5 secs
5 secs
Wie viel Prozent Ihres Jahreseinkommens geben Sie für Technik - Produkte aus ?
2.6 mins
Was spricht für Sie gegen den Kauf eines Neuen Technik Produktes?
3.6 mins
10 secs
5 secs
In welchen der Folgenden Jahre haben Sie sich Technik-Produkte gekauft ?
3.5 mins
7.5 secs
5 secs
Was ist Ihnen beim Kauf eines Technik - Produktes am wichtigsten ?
3.1 mins
5 secs
5 secs
5 secs
Wie zufrieden waren Sie mit Ihrem zuletzt gekauften Technik - Produkt ? (1 = Sehr unzufrieden, 10 = Sehr zufrieden)
4.6 mins
3.5 secs
Nennen Sie die Gründe, die zu Ihrem letzten Kauf eines Technik - Produkts geführt haben.
2.3 mins

Figure 7.2: Performance Perspective indicating mean waiting time between questions and per question

7.2 Organizational Mining

In the context of process mining, organizational mining refers to the organizational perspective. Most event logs also document some kind of information about used resources. Typically, such resources represent who executed a certain activity and are represented as a resource or originator attribute within the event log. As process mining is not limited to control-flow discovery, it is possible to analyze the relation between an activity and its resource, allowing to assign an originator to an activity. Moreover, various social networks may be created by analyzing the relation between different originators during a process. A social network consists of nodes that represent organizational entities and arcs that represent their relationship [96]. An organizational entity may represent a person, certain roles, groups or even departments within an organization. Figure 7.3, which can be found in [14], illustrates a version of a social network in which `Y` is the most important node, indicated by both its size and the weight of 0.98 attached to it. The relationship between `Y` and `X` is stronger than the relationship between `X` and `Z` or `Y` and `Z`, which is again indicated by the thickness of the arc. Since the theory of social network analysis has first been investigated by Jacob Levy Moreno back in 1934 [97], it is certainly not a new topic. Many metrics have already been defined to analyze social networks. An overview about these metrics can be found in [98].

When constructing a social network from event logs, different metrics can be used [99, 100].

1. *Handover of work*

   A handover of work happens if two subsequent activities are completed by two different resources.

2. *Working together*

   Within a working together social network, causal dependencies are not taken into account. Instead the frequency of which resources perform activities in the same case is counted. Individuals that work together more frequently will have a stronger relation than individuals that only rarely work together.

3. *Similar task*

   This metric does not consider how resources work together. Instead it focuses on how similar the executed activities are.

4. *Subcontracting*

   The idea behind this metric is to count how often an activity is executed by a different resource in between two activities from the same resource, which indicates that there is a subcontract for that activity.

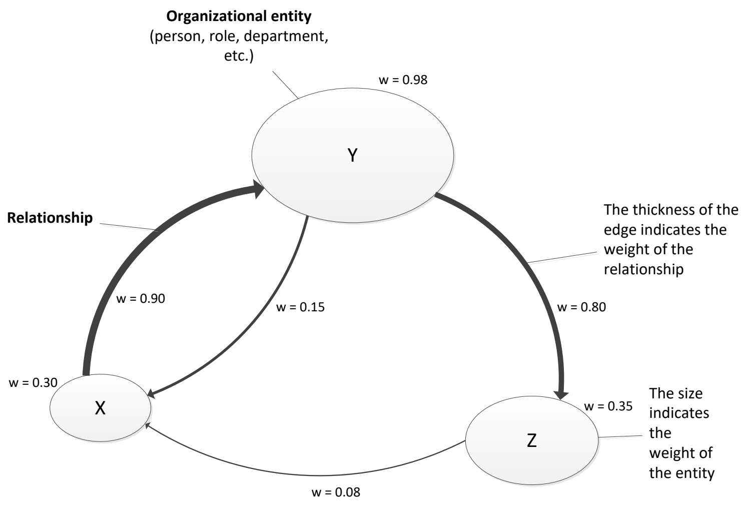

Organizational entity
(person, role, department, etc.)
w = 0.98
Y
Relationship
The thickness of the edge indicates the weight of the relationship
w = 0.90
w = 0.15
w = 0.80
w = 0.30
X
w = 0.35
The size indicates the weight of the entity
Z
w = 0.08

Figure 7.3: Social Network, as found in [14]

7.2.1 Organizational Mining in the Context of Data Collection Scenarios

In the current state, *QuestionSys* does not support behavior similar to different persons executing preceding or subsequent activities, which would allow to construct a social network based on the event log.

However, in future releases it might be possible to extend the questionnaire with different user roles. Multi-User support, as identified in the requirements in Section 3.4.2, would allow the construction of social networks if the different user roles interact within the same questionnaire instance. In the context of medical questionnaires it might be possible to assign specific questions to the role of a doctor, while other questions can only be answered by the patient or their relatives. If this extension is realized, the originator of each question will also be documented in the event log, and it would then be possible to mine a social network from the event log.

7.3 Mining of Decisions

After introducing the time and organizational perspective in the previous sections, this section focuses on the data generated during the process, also called the data perspective. During almost every process execution decisions are made. In real-world processes, this might refer to accepting or rejecting an insurance claim, or the decision if an applicant does get a job offer or not. Based on the decision made, different activities in the process model are executed. Many decisions are influenced by data generated during the process, and the generated data is also captured in the event log. This section describes an approach in which the generated data is used to identify which data element, or combination of data elements, influence a decision. Data mining techniques, like decision trees, are used to combine the data perspective of the event log with the process model. The purpose of doing this is to extract characteristics of each case in the event log and use them to find out in which way they influence certain decisions made during the execution of the process [88].

As a first step, decision points have to be identified. When mining decisions it is irrelevant if the model is a Petri net, an EPC or any other process modeling language, as long as decision points can be identified [101]. A decision point is a point in the process model in which a decision between two or more different activities is made, and only one can be executed. Within Petri nets, this behavior is represented by a place that is an input place for more than one transition. To be able to analyze which path through the model was taken by which instance, the set of possible decisions must be described with respect to the event log. If the process model was discovered by a discovery algorithm, this mapping is already done and is not necessary anymore.

In the next step, each decision point is translated into a classification problem. To do so, structural patterns need to be discovered within the event log. The discipline of machine learning, especially decision trees, may be used to discover such patterns [26]. For each decision point, only those attributes which are known at the time the decision is made are taken into account and are then used as training examples for the decision tree. The further the process is executed, the more training attributes are taken into account. For additional information about decision trees please be referred to [26, 27] or Section 3.3 of this thesis.

Logical expressions can be derived from the created decision trees. If the instance is a leaf node, all of the predicates are fulfilled and therefore connected with an AND connector. If the instance is represented in multiple leaf nodes, the expressions are connected with an OR connector [101]. These logical expressions are used to represent which data influences which decision point, and may yield huge potential when enhancing a process model.

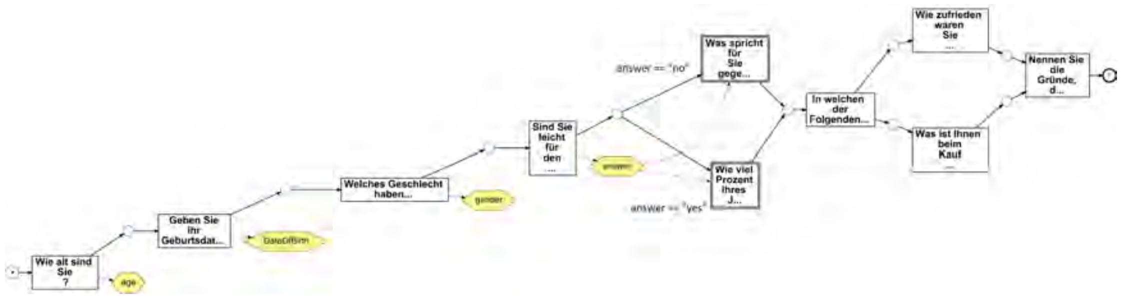
7.3.1 Decision Mining in the Context of Data Collection Scenarios

As the underlying process model is strictly specified within *QuestionSys*, Decision Mining might not generate new information at first glance, as it will always rediscover the guards specified during questionnaire creation.

The decision tree used by the Decision Miner takes all data into account that was generated before the decision point. This might lead to the discovery of different guards than the specified ones which indicates that either the guards are not ideal, or a new relationship within the questionnaire was found.

Figure 7.4 displays the result of applying the *Decision-tree mining* plugin in *ProM* [102] to an event log similar to the one used in Section 7.1. The event log has been modified to also capture the answers to each question prior to the decision point. The answers are documented as data elements within the event log and displayed as yellow hexagons in the process model. This enhances the process model and allows to identify at which point in the questionnaire certain data is generated and which data influences which decision.

In Figure 7.4 a decision is made after the fourth question is answered. The Decision Miner indicates that the answer given to that question (data element answer) determines which of the two questions is answered next. If the participant answered with *"yes"*, the next question is about the percentage of income spent for new electronic products, whereas if the answer is *"no"*, the participant is asked to provide reasons against purchasing new electronic products. This decision is directly derived from the generated decision tree.


Wie alt sind Sie ?
age
Geben Sie Ihr Geburtsdat...
DateDiffBirth
Welches Geschlecht haben...
gender
Sind Sie leicht für den ...
answer
answer == "no"
Was spricht für Sie gege...
answer == "yes"
Wie viel Prozent Ihres J...
In welchen der Folgenden...
Wie zufrieden waren Sie ...
Was ist Ihnen beim Kauf ...
Nennen Sie die Gründe, d...
Figure 7.4: Process model enriched with data indication decision-relevant data

The Decision Miner plugin in *ProM* also has the possibility to specify which data objects should be considered as input variables for the decision tree. By removing the data element *"answer"* from the considered variables, the Decision Miner is able to find a new pattern in the data.

As displayed in Figure 7.5, the new guard is the data element *"gender"*. Therefore, it may be inferred that women are less likely to purchase new electronic products, whereas men are easier to convince to purchase electronic products.

The example provided in Figure 7.5 is just one way of applying the *Decision-tree mining* plugin included in *ProM* to generate new insights. Based on different questionnaires the approach may lead to more new information.

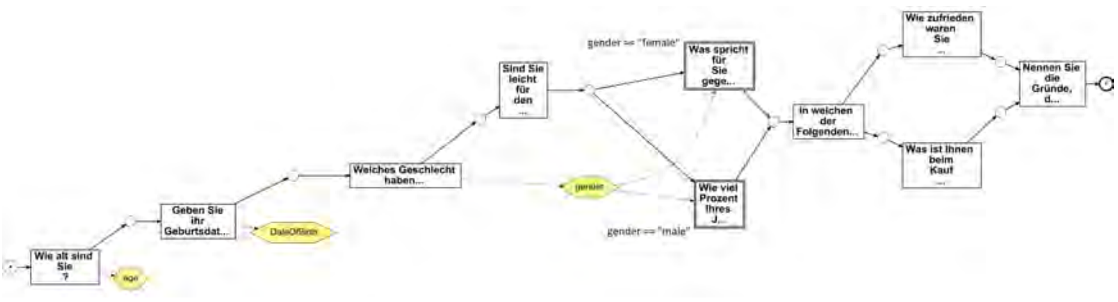

Wie alt sind Sie ?
age
Geben Sie ihr Geburtsdat...
DateOfBirth
Welches Geschlecht haben...
Sind Sie leicht für den ...
gender == "female"
gender
gender == "male"
Was spricht für Sie gege...
Wie viel Prozent Ihres J...
In welchen der Folgenden...
Wie zufrieden waren Sie ...
Was ist Ihnen beim Kauf ...
Nennen Sie die Gründe, d...

Figure 7.5: Process model enriched with data indication decision-relevant data, excluding the specified guard

Moreover, this approach is not limited to the answers given as data objects. Each question is able to generate multiple data objects, which is displayed in Figure 7.6. Additional information gathered from e.g. vital sensors or the microphone of the smartphone, may also be used as input variables for the decision tree [36]. This would then allow to use this data in addition to the answers to get an even more detailed view of the data perspective.

Enriching an event log with such data allows to include this data in the decision tree and may identify unknown relationships within the questionnaire. The provided data should only be seen as examples of possible data that may be collected as there is no limit to the various types of data that may be collected additionally.

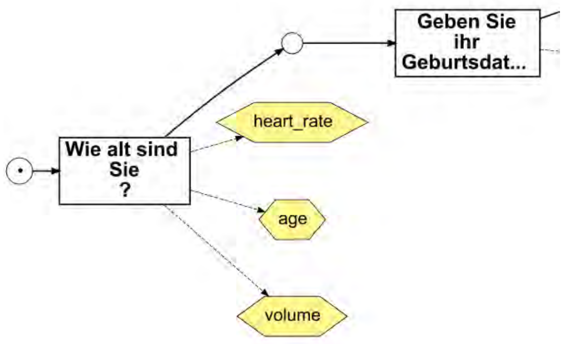


Figure 7.6: Example of multiple data objects generated by a question

8 Conclusion

This final chapter summarizes the thesis. A summary of the contribution towards the objective is provided in Section 8.1. Section 8.2 provides an outlook on possible future use cases of process mining in the context of questionnaires.

8.1 Contribution

In the course of this thesis, an overview of process mining was developed. After establishing a basic understanding of process mining, various process discovery algorithms are introduced. Existing properties such as short loops or non-free choice constructs are put in the context of questionnaires, and possible novel analysis based on these properties are presented. In addition, different process discovery algorithms are analyzed based on these properties and their applicability in the context of data collection scenarios is evaluated.

Process mining is in fact not limited to process discovery, and the second form of process mining, conformance checking, may also be applied to questionnaires. Different algorithms for conformance checking and their applicability in the context of questionnaires are presented. Footprint comparison may be used to follow up on structural changes between different versions of a questionnaire and help to quantify the effectiveness of these changes. Other conformance checking techniques can be used to identify at which point in the questionnaire participants drop out, and this information can then be used to improve the questionnaire in further releases. Additionally, an approach based on Linear Temporal Logic is presented which offers an intuitive way of testing assumptions by filtering the event log based on predefined formulas. The separated event logs may then be used as input for other process mining algorithms to further investigate certain properties.

Event logs offer the possibility to document additional data collected from questionnaires. This data can be used to enrich questionnaires in order to represent additional perspectives and possibly discover new relationships within the questionnaire. In this thesis, the temporal perspective is used to analyze the time needed to answer each question and the questionnaire as a whole allowing for novel insights compared to paper-based questionnaires. The data perspective is used to enrich the questionnaire with data generated during its execution. Data collected by a questionnaire can be used to identify which decision within the questionnaire is influenced by which data. The data used in Chapter 7 of this thesis only represents a fraction of the possible extensions as questionnaires are able to generate a plethora of different data. Depending on the questionnaire, the generated data may vary and is not restricted to answers given. Sensors within a smartphone may be used to measure the volume or the heart rate when answering certain questions. Using this kind of data extends the capabilities of analyzing a questionnaire substantially.

This thesis shows that process mining can be applied successfully in the context of data collection scenarios. The event logs used in this thesis may also be used as a guideline to create event logs for process mining in the context of *QuestionSys*. Not only does this thesis provide an overview of different process mining algorithms, but it may also be used to improve the effectiveness of questionnaire analysis in the context of process-oriented questionnaires.

8.2 Outlook

As described in Section 1.2, the event logs used in this work are generated artificially. A possible next step would be applying the presented process mining algorithms to event logs from real-life studies performed with *QuestionSys*. Since process mining is a relatively young research discipline, new and better algorithms are developed across all forms rather quickly. These new algorithms may also be beneficial in the context of questionnaires.

Even if this thesis used an offline approach for process mining, the algorithms may also be used online. To use process mining online, operational support capabilities need to be included in existing information systems. This may be a challenging task, but it would make real-time, instance-specific recommendations and predictions possible. In the context of clinical questionnaires, a doctor may be informed based on the answers given by the patient even before the patient completed the questionnaire. Doctors may be supported with case-specific predictions derived from previous questionnaires.

Even if process discovery is the most prominent form of process mining and a lot of research is done in this area, it is still very difficult to provide a general rule on which algorithm to use. Each event log has different properties and most process discovery algorithms are specially tailored towards a specific scenario. In the area of process discovery, improvements to the representational bias are necessary to improve the efficiency of process discovery algorithms and the quality of their results.

Both conformance checking and enhancement are also able to generate novel information. Unfortunately, most of the commercial process mining tools do not support conformance checking and enhancement to a satisfying degree. Better and more efficient techniques need to be developed.

Event logs are crucial for process mining because process mining results are heavily dependent on the quality of the underlying event log. Data or behavior that is not documented in the event log cannot be used to derive relations or enhance the process model with additional perspectives. Unfortunately, event logs are often scattered, in an unusable format or events are simply overwritten which makes process mining unnecessary complicated or even impossible. Among other things, the increasing awareness of process mining should be used to increase the quality of event logs.

Bibliography

[1] Schobel, J.: QuestionSys - A Generic and Flexible Questionnaire System Enabling Process-Driven Mobile Data Collection. https://www.uni-ulm.de/in/iui-dbis/forschung/laufende-projekte/questionsys/ (2012) last accessed: 02.09.2018.

[2] Schobel, J., Pryss, R., Schickler, M., Reichert, M.: A Lightweight Process Engine for Enabling Advanced Mobile Applications. In: 24th International Conference on Cooperative Information Systems (CoopIS 2016). Number 10033 in LNCS, Springer (2016) 552–569

[3] Schobel, J., Schickler, M., Pryss, R., Maier, F., Reichert, M.: Towards Process-Driven Mobile Data Collection Applications: Requirements, Challenges, Lessons Learned. In: 10th Int'l Conference on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps. (2014) 371–382

[4] Mans, R.S., Schonenberg, M., Song, M., van der Aalst, W.M.P., Bakker, P.J.: Application of Process Mining in Healthcare - A Case Study in a Dutch Hospital. In: International joint conference on biomedical engineering systems and technologies, Springer (2008) 425–438

[5] Lamine, E., Fontanili, F., Di Mascolo, M., Pingaud, H.: Improving the Management of an Emergency Call Service by Combining Process Mining and Discrete Event Simulation Approaches. In: Working Conference on Virtual Enterprises, Springer (2015) 535–546

[6] van der Aalst, W.M.P., Reijers, H.A., Weijters, A.J.M.M., van Dongen, B.F., De Medeiros, A.K.A., Song, M., Verbeek, H.: Business process mining: An industrial application. Information Systems **32** (2007) 713–732

[7] Bala, S., Mendling, J.: Monitoring the Software Development Process with Process Mining. In: International Symposium on Business Modeling and Software Design, Springer (2018) 432–442

[8] Andrews, R., Wynn, M.T., Vallmuur, K., ter Hofstede, A.H., Bosley, E., Elcock, M., Rashford, S.: Pre-hospital Retrieval and Transport of Road Trauma Patients in Queensland: A Process Mining Analysis. (2018)

[9] Rojas, E., Munoz-Gama, J., Sepúlveda, M., Capurro, D.: Process mining in healthcare: A literature review. Journal of Biomedical Informatics **61** (2016) 224–236

[10] Mans, R., Schonenberg, H., Leonardi, G., Panzarasa, S., Cavallini, A., Quaglini, S., van der Aalst, W.M.P.: Process Mining Techniques: an Application to Stroke Care. Studies in Health Technology and Informatics **136** (2008) 573–8

[11] Rozinat, A., de Jong, I.S., Günther, C., van der Aalst, W.M.P.: Process Mining Applied to the Test Process of Wafer Steppers in ASML. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) **39** (2009) 474–479

[12] Guenther, C.W., Rinderle-Ma, S., Reichert, M., van der Aalst, W.M., Recker, J.: Using process mining to learn from process changes in evolutionary systems. Int'l Journal of Business Process Integration and Management, Special Issue on Business Process Flexibility **3** (2008) 61–78

[13] van der Aalst, W.M.P.: Extracting Event Data from Databases to Unleash Process Mining. In: BPM. Springer (2015) 105–128

[14] van der Aalst, W.M.P.: Process Mining: Data Science in Action, Second Edition. Springer (2016)

[15] Günther, C.W., Verbeek, E.: Xes Standard Definition. Fluxicon Process Laboratories **13** (2009) 14

[16] van Dongen, B.F., van der Aalst, W.M.P.: A Meta Model for Process Mining Data. In: In Proceedings of the CAiSE WORKSHOPS. (2005) 309–320

[17] van der Aalst, W.M.P., van Dongen, B.F.: Discovering Petri Nets from Event Logs. In: Transactions on Petri Nets and Other Models of Concurrency VII. Springer (2013) 372–422

[18] van der Aalst, W.M.P.: The Application of Petri Nets to Workflow Management. Journal of Circuits, Systems, and Computers **8** (1998) 21–66

[19] Petri, C.A.: Kommunikation mit Automaten. PhD thesis, Universität Hamburg (1962)

[20] Murata, T.: Petri nets: Properties, Analysis and Applications. Proceedings of the IEEE **77** (1989) 541–580

[21] van der Aalst, W.M.P.: A class of Petri nets for modeling and analyzing business processes. Computing Science Reports **95** (1995) 26

[22] van der Aalst, W.M.P.: Verification of Workflow Nets. In: International Conference on Application and Theory of Petri Nets. Lecture Notes in Computer Science, Springer (1997) 407–426

[23] van der Aalst, W.M.P., van Hee, K.M., ter Hofstede, A.H.M., Sidorova, N., Verbeek, H.M.W., Voorhoeve, M., Wynn, M.T.: Soundness of workflow nets: classification, decidability, and analysis. Formal Aspects of Computing **23** (2011) 333–363

[24] Verbeek, H.M.W., Wynn, M.T., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Reduction rules for reset/inhibitor nets. Journal of Computer and System Sciences **76** (2010) 125–143

[25] Zimmermann, A.: Stochastic Discrete Event Systems. Springer (2007)

[26] Mitchell, T.M.: Machine learning. McGraw Hill series in computer science. McGraw-Hill (1997)

[27] Witten, I.H., Frank, E., Hall, M.A., Pal, C.J.: Chapter 3 - Output: Knowledge Representation. Third edition edn. Morgan Kaufmann, Boston (2011)

[28] Quinlan, J.R.: Induction of Decision Trees. Machine learning **1** (1986) 81–106

[29] Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1993)

[30] Mannhardt, F., De Leoni, M., Reijers, H.A.: The Multi-perspective Process Explorer. In: Proceedings of the Demo Session of the 13th International Conference on Business Process Management (BPM 2015, Innsbruck, Austria). (2015) 130–134

[31] Digital Porpoise LLS - dpwp2015: Which decision tree format is easier for humans? http://digitalporpoise.com/2016/03/which-decision-tree-format-is-easier-for-humans/ (2016) last accessed: 16.09.2018.

[32] Schobel, J., Pryss, R., Schlee, W., Probst, T., Gebhardt, D., Schickler, M., Reichert, M.: Development of Mobile Data Collection Applications by Domain Experts: Experimental Results from a Usability Study. In: 29th International Conference on Advanced Information Systems Engineering (CAiSE 2017). Number 10253 in LNCS, Springer (2017) 60–75

[33] Ruf-Leuschner, M., Pryss, R., Liebrecht, M., Schobel, J., Spyridou, A., Reichert, M., Schauer, M.: Preventing further trauma: KINDEX mum screen - assessing and reacting towards psychosocial risk factors in pregnant women with the help of smartphone technologies. In: XIII Congress of European Society of Traumatic Stress Studies (ESTSS) Conference. (2013) 70–70

[34] Isele, D., Ruf-Leuschner, M., Pryss, R., Schauer, M., Reichert, M., Schobel, J., Schindler, A., Elbert, T.: Detecting adverse childhood experiences with a little help from tablet computers. In: XIII Congress of European Society of Traumatic Stress Studies (ESTSS) Conference. (2013) 69–70

[35] Schobel, J., Schickler, M., Pryss, R., Reichert, M.: Process-Driven Data Collection with Smart Mobile Devices. In: 10th International Conference on Web Information Systems and Technologies (Revised Selected Papers). Number 226 in LNBIP. Springer (2015) 347–362

[36] Schobel, J., Schickler, M., Pryss, R., Nienhaus, H., Reichert, M.: Using Vital Sensors in Mobile Healthcare Business Applications: Challenges, Examples, Lessons Learned. In: 9th Int'l Conference on Web Information Systems and Technologies (WEBIST 2013), Special Session on Business Apps. (2013) 509–518

[37] Schobel, J., Pryss, R., Wipp, W., Schickler, M., Reichert, M.: A Mobile Service Engine Enabling Complex Data Collection Applications. In: 14th International Conference on Service Oriented Computing (ICSOC 2016). Number 9936 in LNCS (2016) 626–633

[38] van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow mining: A survey of issues and approaches. Data & Knowledge Engineering **47** (2003) 237–267

[39] van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H.M.W., Weijters, A.J.M.M., van der Aalst, W.M.P.: The ProM Framework: A New Era in Process Mining Tool Support. In: Applications and Theory of Petri Nets 2005, Springer Berlin Heidelberg (2005) 444–454

[40] Verbeek, H.M.W., Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: Prom 6: The Process Mining Toolkit. In: Proceedings of the Business Process Management 2010 Demonstration Track (Hoboken NJ, USA, September 14-16, 2010). (2010) 34–39

[41] Günther, C.W., Rozinat, A.: Disco: Discover Your Processes. BPM (Demos) **940** (2012) 40–44

[42] Carmona Vargas, J., Solé, M.: PMLAB: An Scripting Environment for Process Mining. In: Proceedings of the BPM Demo Sessions 2014: Co-located with the 12th International Conference on Business Process Management (BPM 2014) Eindhoven, The Netherlands, September 10, 2014, CEUR-WS.org (2014) 16–20

[43] vanden Broucke, S., De Weerdt, J., Vanthienen, J., Baesens, B.: A Comprehensive Benchmarking Framework (CoBeFra) for Conformance Analysis between Procedural Process Models and Event Logs in ProM. In: 2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM), IEEE (2013) 254–261

[44] van der Aalst, W.M.P., Adriansyah, A., de Medeiros, A.K.A., Arcieri, F., Baier, T., et al.: Process Mining Manifesto. In: Business Process Management Workshops (1). Volume 99 of Lecture Notes in Business Information Processing., Springer (2011) 169–194

[45] Dhar, V.: Data science and prediction. Communications of the ACM **56** (2013) 64–73

[46] van der Aalst, W.M.P., Schonenberg, M.H., Song, M.: Time Prediction Based on Process Mining. Information systems **36** (2011) 450–475

[47] Netjes, M., Reijers, H.A., van der Aalst, W.M.P.: Supporting the Full BPM Life-Cycle using Process Mining and Intelligent Redesign. In: Contemporary issues in database design and information systems development. IGI Global (2007) 100–132

[48] van der Aalst, W.M.P., La Rosa, M., Santoro, F.M.: Business Process Management. Business & Information Systems Engineering (2016)

[49] Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A., et al.: Fundamentals of Business Process Management. Volume 1. Springer (2013)

[50] Mendling, J., Neumann, G., van der Aalst, W.M.P.: Understanding the Occurrence of Errors in Process Models based on Metrics. In: On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS, Springer Berlin Heidelberg (2007) 113–130

[51] Rozinat, A., De Medeiros, A.K.A., Günther, C.W., Weijters, A.J.M.M., van der Aalst, W.M.P.: Towards an Evaluation Framework for Process Mining Algorithms. BPM Center Report BPM-07-06, BPMcenter.org **123** (2007) 142

[52] Carmona, J., Cortadella, J., Kishinevsky, M.: A Region-Based Algorithm for Discovering Petri Nets from Event Logs. In: Business Process Management, Springer Berlin Heidelberg (2008) 358–373

[53] van Dongen, B.F., van der Aalst, W.M.P.: Multi-phase Process Mining: Building Instance Graphs. In: ER. Lecture Notes in Computer Science, Springer (2004) 362–376

[54] Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P. In: Process and Deviation Exploration with Inductive Visual Miner. CEUR-WS.org (2014) 46–50

[55] Tiwari, A., Turner, C.J., Majeed, B.: A review of business process mining: state-of-the-art and future trends. Business Process Management Journal **14** (2008) 5–22

[56] van der Aalst, W.M.P., Weijters, A.J.M.M.: Process mining: a research agenda. Computers in Industry **53** (2004) 231–244

[57] van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow Mining: Discovering Process Models from Event Logs. IEEE Transactions on Knowledge and Data Engineering **16** (2004) 1128–1142

[58] Alves de Medeiros, A.K., Van Dongen, B.F., van der Aalst, W.M.P., Weijters, A.J.M.M.: Process Mining: Extending the α-algorithm to Mine Short Loops. BETA publicatie:working papers. Technische Universiteit Eindhoven (2004)

[59] van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow Mining: Which processes can be rediscovered? Technical report, BETA Working Paper Series, WP 74, Eindhoven University of Technology, Eindhoven (2002)

[60] de Medeiros, A.K.A., van der Aalst, W.M.P., Weijters, A.J.M.M.: Workflow Mining: Current Status and Future Directions. In: CoopIS/DOA/ODBASE. Volume 2888 of Lecture Notes in Computer Science., Springer (2003) 389–406

[61] Guo, Q., Wen, L., Wang, J., Yan, Z., Philip, S.Y.: Mining Invisible Tasks in Non-free-choice Constructs. In: Business Process Management, Springer (2015) 109–125

[62] Wen, L., Wang, J., van der Aalst, W.M.P., Huang, B., Sun, J.: A Novel Approach for Process Mining Based on Event Types. Journal of Intelligent Information Systems **32** (2009) 163–190

[63] de Medeiros, A.K.A., van Dongen, B.F., van der Aalst, W.M.P., Weijters, A.J.M.M.: Process Mining for Ubiquitous Mobile Systems: An Overview and a Concrete Algorithm. In: Ubiquitous Mobile Information and Collaboration Systems, Springer Berlin Heidelberg (2004) 151–165

[64] van der Aalst, W.M.P., van Dongen, B.F.: Discovering Workflow Performance Models from Timed Logs. In: Engineering and Deployment of Cooperative Information Systems. Springer Berlin Heidelberg (2002) 45–63

[65] Aleem, S., Capretz, L.F., Ahmed, F.: Business process mining approaches: A relative comparison. CoRR (2015)

[66] Desel, J., Esparza, J.: Free Choice Petri Nets. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press (1995)

[67] Wen, L., van der Aalst, W.M.P., Wang, J., Sun, J.: Mining process models with non-free-choice constructs. Data Mining and Knowledge Discovery **15** (2007) 145–180

[68] Wen, L., Wang, J., Sun, J.: Detecting Implicit Dependencies Between Tasks from Event Logs. In: Frontiers of WWW Research and Development - APWeb 2006, Springer Berlin Heidelberg (2006) 591–603

[69] Weijters, A.J.M.M., Ribeiro, J.T.S.: Flexible Heuristics Miner (FHM). In: Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on, IEEE (2011) 310–317

[70] van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.: Causal Nets: A Modeling Language Tailored Towards Process Discovery. In: International conference on concurrency theory, Springer (2011) 28–42

[71] Weijters, A.J.M.M., van der Aalst, W.M.P., De Medeiros, A.K.A.: Process Mining with the HeuristicsMiner Algorithm. Technische Universiteit Eindhoven, Tech. Rep. WP **166** (2006) 1–34

[72] Weijters, A.J.M.M., van der Aalst, W.M.P.: Rediscovering Workflow Models from Event-Based Data using Little Thumb. Integrated Computer-Aided Engineering **10** (2003) 151–162

[73] de Medeiros, A.K.A., Weijters, A.J.M.M., van der Aalst, W.M.P.: Genetic Process Mining: A Basic Approach and Its Challenges. In: Business Process Management Workshops, Springer Berlin Heidelberg (2006) 203–215

[74] Günther, C.W., van der Aalst, W.M.P.: Fuzzy Mining - Adaptive Process Simplification Based on Multi-perspective Metrics. In: BPM. Lecture Notes in Computer Science, Springer (2007) 328–343

[75] Günther, C.W.: Process Mining in Flexible Environments. PhD thesis (2009)

[76] Gupta, E.P.: Process Mining A Comparative Study. International Journal of Advanced Research in Computer and Communications Engineering **3** (2014) 5

[77] Darwin, C.: On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life, 1859. Routledge (2004)

[78] Medeiros, A.K.A., Weijters, A.J.M.M., van der Aalst, W.M.P.: Using Genetic Algorithms to Mine Process Models: Representation, Operators and Results. BETA publicatie:working papers. Technische Universiteit Eindhoven (2004)

[79] de Medeiros, A.K.A., Weijters, A.J.M.M., van der Aalst, W.M.P.: Genetic process mining: an experimental evaluation. Data Mining and Knowledge Discovery **14** (2007) 245–304

[80] Alves de Medeiros, A.K.: Genetic Process Mining. PhD thesis (2006)

[81] van der Aalst, W.M.P., de Medeiros, A.K.A., Weijters, A.J.M.M.: Genetic Process Mining. In: International Conference on Application and Theory of Petri Nets, Springer (2005) 48–69

[82] van der Aalst, W.M.P.: Process Mining: Overview and opportunities. ACM Transactions on Management Information Systems **3** (2012) 1–17

[83] Weidlich, M., van der Werf, J.M.: On Profiles and Footprints - Relational Semantics for Petri Nets. In: International Conference on Application and Theory of Petri Nets and Concurrency, Springer (2012) 148–167

[84] Molka, T., Redlich, D., Drobek, M., Caetano, A., Zeng, X.J., Gilani, W.: Conformance Checking for BPMN-Based Process Models. In: Proceedings of the 29th Annual ACM Symposium on Applied Computing, ACM (2014) 1406–1413

[85] van Dongen, B.F., Mendling, J., van der Aalst, W.M.P.: Structural Patterns for Soundness of Business Process Models. In: EDOC, IEEE Computer Society (2006) 116–128

[86] van Dongen, B., Dijkman, R., Mendling, J.: Measuring Similarity between Business Process Models. In: Advanced Information Systems Engineering, Springer Berlin Heidelberg (2008) 450–464

[87] Rozinat, A., van der Aalst, W.M.P.: Conformance Checking of Processes Based on Monitoring Real Behavior. Information Systems **33** (2008) 64–95

[88] Rozinat, A.: Process Mining: Conformance and Extension. PhD thesis (2010)

[89] De Medeiros, A.K.A., van der Aalst, W.M.P., Weijters, A.J.M.M.: Quantifying process equivalence based on observed behavior. Data & knowledge engineering **64** (2008) 55–74

[90] Rozinat, A., van der Aalst, W.M.P.: Conformance Testing: Measuring the Fit and Appropriateness of Event Logs and Process Models. In: International Conference on Business Process Management, Springer (2005) 163–176

[91] van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.: Replaying History on Process Models for Conformance Checking and Performance Analysis. Wiley Int. Rev. Data Min. and Knowl. Disc. **2** (2012) 182–192

[92] Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P.: Conformance Checking Using Cost-Based Fitness Analysis. 2011 IEEE 15th International Enterprise Distributed Object Computing Conference (2011) 55–64

[93] van der Aalst, W.M.P., de Beer, H.T., van Dongen, B.F.: Process Mining and Verification of Properties: An Approach Based on Temporal Logic. In: Proceedings of the 2005 Confederated International Conference on On the Move to Meaningful Internet Systems, Springer Verlag (2005) 130–147

[94] Clarke, Jr., E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge, MA, USA (1999)

[95] Maggi, F.M., Westergaard, M., Montali, M., van der Aalst, W.M.P.: Runtime Verification of LTL-Based Declarative Process Models. In: International Conference on Runtime Verification, Springer (2011) 131–146

[96] van der Aalst, W.M.P., Song, M.: Mining Social Networks: Uncovering Interaction Patterns in Business Processes. In: Business Process Management, Springer Berlin Heidelberg (2004) 244–260

[97] Moreno, J.L.: Who shall survive?: A new approach to the problem of human interrelations. Nervous and mental disease monograph series, no 58. (1934)

[98] Wasserman, S., Faust, K.: Social Network Analysis: Methods and Applications. Structural Analysis in the Social Sciences. Cambridge University Press (1994)

[99] van der Aalst, W.M.P., Reijers, H.A., Song, M.: Discovering social networks from event logs. Computer Supported Cooperative Work (CSCW) **14** (2005) 549–593

[100] Song, M., van der Aalst, W.M.P.: Towards Comprehensive Support for Organizational Mining. Decision Support Systems **46** (2008) 300–317

[101] Rozinat, A., van der Aalst, W.M.P.: Decision Mining in Business Processes. BETA publicatie:working papers. Technische Universiteit Eindhoven (2006)

[102] Rozinat, A., van der Aalst, W.M.P.: Decision Mining in ProM. In: Business Process Management. Lecture Notes in Computer Science, Springer (2006) 420–425

List of Figures

List of Tables

Name: Marius Breitmayer Matriculation number: 805851

Honesty disclaimer

I hereby affirm that I wrote this thesis independently and that I did not use any other sources or tools than the ones specified.

Ulm, ...

Marius Breitmayer