Empirically Evaluating Process Mining Algorithms

Towards Closing the Methodological Gap

Toon Jouck

UHasselt - Hasselt University, Faculty of Business Economics

August 2018

Acknowledgements

This thesis is the product of four years of research at UHasselt. It did not feel as a long and tiring journey to get to this point, except for the last months maybe. However, there were many ups and downs along the way. Some papers were accepted quickly, others took ages to get through all the review rounds or were rejected. Collaborations were initiated and most of them have led to results which I am proud of, but more importantly, they meant meeting new people and produced inspiring discussions. Of course, none of these things would have happened without the aid of other people who helped me to complete this thesis and overcome the obstacles along the way. Therefore, I want to thank those people.

First of all, I would like to thank each of my jury members for his/her interesting questions and remarks which contributed a lot to this thesis. Prof. dr. Andrea Burattin, Prof. dr. Josep Carmona, Prof. dr. Jochen De Weerdt, Prof. dr. Koen Vanhoof, thank you all for your useful suggestions. I really appreciate the constructive feedback you provided to improve this thesis. Thank you, Prof. dr. Mieke Jans for your guidance and support during my four years here. I really appreciated the collaboration for the course of business process analytics, even though it meant doing a few extra tasks every now and then. Thank you, Prof. dr. Massimiliano de Leoni for the pleasant research collaboration. You provided that spark that was necessary for our research project to succeed and thank you for being such a warm person in general. I absolutely enjoyed going out for a beer at the BPM conference or in Eindhoven. A special thanks goes to my supervisor Prof. dr. Benoît Depaire. You gave me the opportunity to start this PhD. Without your intelligent counseling and infinite optimism, I would have never been able to complete it. The hours of discussions we had contributed immensely to this thesis, but also have helped me in other parts of my life.

Furthermore, I want to thank all of my colleagues of Quantitative Methods. Mentioning each one of you and their respective contribution would take a lot of space and time and would most likely fail in being exhaustive. So please forgive me for being lazy, but know that I am indebted to all of you. Thanks to you, I could count the number of days I did not want to go to work on one hand. Each of you has contributed in his or her own way by listening, supporting, and philosophizing about anything you can imagine. I greatly appreciate it that you all sustained my endless series of silly jokes and puns. Moreover, I would like to thank the people of the Information Systems group at TUe for having me there on an inspiring research visit. Special thanks goes to Alfredo who worked with me and helped me enormously with my research code.

I also would like to thank my family and friends because I could lean on them in difficult times and for the distraction they offered when needed. There is really no greater thing than being part of a large family with three brothers and a sister together with their spouses and children. I am eternally grateful for their advice and joy they bring in my life. A very special thanks goes to my mother and father who encouraged me to embark on the PhD journey and who stuck with me through thick and thin. Furthermore, I would like to express my gratitude for my friends of the "bierpuntcrew" and of my basketball team for all the joy that kept me motivated. Also thank you Marijke and Johan for letting me use your (internet) facilities. Last but not least, thank you Lore for being by my side no matter what. I do not know what I would do without you. You contributed to this PhD in many ways, not only by doing all the ironing and cleaning during the past two years. I will not do another PhD, I promise.

Toon Jouck
August 2018

Samenvatting

Om competitief te blijven op een globale en snel veranderende markt, hebben bedrijven hun focus verlegd van de producten en diensten naar de manier waarop deze gemaakt en geleverd worden aan de klant. Een gevolg daarvan is dat het beheren van de bedrijsprocessen belangrijker is geworden. Bedrijven hebben steeds meer en meer data verzameld over de uitvoering van deze bedrijfsprocessen ondersteund door informatiesystemen. Dit heeft geleid tot een explosie van beschikbare procesdata.

Process mining technieken bieden de mogelijkheid om onontgonnen kennis uit zulke procesdata, ook wel event logs genoemd, te halen. Deze technieken reiken praktische inzichten en ideeën aan met het doel bedrijfsprocessen te verbeteren en hun performantie te verhogen. Het startpunt van process mining is het ontdekken van een procesmodel uit de event log, ook wel process discovery genoemd. Het ontdekte model biedt een objectieve weergave van de realiteit door de volgorde van de verschillende procesactiviteiten te visualiseren. Met behulp van het ontdekte model kan je twee andere types van process mining technieken toepassen, namelijk conformance checking en enhancement. Conformance checking technieken sporen afwijkingen op tussen een event log en het ontdekte model. Enhancement technieken voegen informatie toe aan het ontdekte model op basis van de gegevens in de event log. Voorbeelden hiervan zijn kosten, eigenschappen van de procesinstantie en klantgegevens. Een essentieel onderdeel van procesmodellen zijn de beslissingspunten waar bepaald wordt welk pad de procesinstantie doorheen het proces volgt. Zo kan een bedrijf bijvoorbeeld korting toekennen aan loyale klanten. Het ontdekte model bevat zulke informatie niet. Met behulp van decision mining technieken kan je op basis van de informatie in de event log leren hoe de beslissingen in het ontdekte

model genomen worden.

Het stijgende aantal process discovery en decision mining technieken heeft er toe geleid dat het onderzoek omtrent het empirisch evalueren van deze technieken meer aandacht heeft gekregen. Deze evaluaties hebben als doel om inzichten te leveren over welke technieken goed werken bij welke soorten procesgedrag. Een empirische evaluatie van process discovery/decision mining technieken vereist de volgende vier stappen: het bepalen van het doel van de evaluatie, het selecteren van de procesdata, het kiezen van een geschikte kwaliteitsmaatstaf en het toepassen van de correcte statistische test. Elk van deze vier stappen biedt onopgeloste vraagstukken die het onderzoeksdomein verhinderen inzichten te verkrijgen in de sterktes en zwaktes van process discovery en decision mining technieken. Deze thesis spitst zich toe op de uitdagingen rond de selectie van procesdata, kwaliteitsmeting en statistische tests voor process discovery. Bovendien pakt deze thesis ook de uitdagingen rond procesdata en kwaliteitsmeting voor decision mining evaluatie aan.

Het selecteren van de procesdata biedt de eerste uitdaging voor process discovery evaluatie. Geen enkele van de bestaande evaluatiemethoden specifieert een methodologie voor het kiezen van de geschikte procesdata voor het evalueren van process discovery technieken. Bovendien bevat de bestaande verzameling van reële event logs, die vaak gebruikt wordt voor evaluatie, geen referentiemodellen waardoor ze niet toelaat om statistisch significante conclusies te veralgemenen naar een procespopulatie. Anderzijds zijn bestaande artificiële datageneratoren beperkt in de proceskarakteristieken die ze genereren en garanderen ze geen correct experimenteel ontwerp zodat de geldigheid van de statistische conclusies niet zeker is. De tweede uitdaging omvat het meten van de kwaliteit van de ontdekte modellen. De huidige evaluatiemethoden vertrouwen op maatstaven die sterk gelinkt zijn aan de modelnotatie en hierdoor de kwaliteitsresultaten beïnvloeden. De finale uitdaging betreft de statistische tests die uitgevoerd worden om algemene conclusies te trekken op basis van de resultaten. De huidige evaluatiemethoden gebruiken niet-willekeurige steekproeven waarvan men de populatie niet kan achterhalen en dus de resultaten niet veralgemeend kunnen worden naar de populatie.

De belangrijkste uitdaging bij het evalueren van decision mining technieken is het ontbreken van een standaardprocedure. Als gevolg daarvan zijn er maar enkele empirische evaluaties uitgevoerd die problemen ondervonden met de selectie van procesdata en het meten van de kwaliteit. Deze evaluaties gebruikten kleine niet-willekeurige steekproeven die niet veralgemeend kunnen worden naar een populatie. Bestaande artificiële datageneratoren zijn niet ontworpen voor het evalueren van decision mining technieken en bieden geen oplossing voor de bestaande uitdagingen. Bovendien hebben de huidige evaluatiemethoden verschillende kwaliteitsmaatstaven gebruikt die niet volledig objectief zijn. Tot slot heeft de beperking tot kleine steekproeven geleid tot het ontbreken van statistische analyses in de bestaande evaluaties.

Zolang het onderzoeksdomein de bestaande uitdagingen gerelateerd aan de evalatie van process discovery en decision mining technieken niet aanpakt, zal er geen consensus zijn over de kwaliteit van de bestaande technieken. Daarom is het hoofddoel van deze thesis het ontwerpen van empirische evaluatieprocedures voor zowel process discovery als decision mining die een objective vergelijking en veralgemening van de resultaten toelaten. Het hoofddoel is verder opgedeeld in drie onderzoeksdoelen.

Het eerste onderzoeksdoel omvat de *Generating artificial Event Data (GED)* methodologie voor het genereren van willekeurige procesmodellen en event logs voor empirische evaluatie van process discovery en decision mining technieken. De GED methodologie start met het definiëren van de procesmodelpopulatie. Deze definitie specifieert de procespatronen die de modellen in de populatie karakteriseren. In een volgende stap wordt een willekeurige steekproef bestaande uit procesmodellen getrokken uit de populatie. Deze steekproef wordt dan gesimuleerd in willekeurige event logs. De *Process Tree and Log Generator (PTandLogGenerator)* voorziet de nodige algoritmes en ondersteuning om de GED methodologie te implementeren en te automatiseren. De nieuwe algoritmes maken het mogelijk om procespatronen, i.e. langetermijnafhankelijkheden, meerkeuze en gedupliceerde activiteiten, te introduceren in de gegenereerde modellen die niet mogelijk waren in bestaande datageneratoren. De evaluatie van de PTandLogGenerator toont aan dat deze effectief de GED methodologie ondersteunt en leidt tot nieuwe inzichten over process discovery technieken. Bovendien maakt de uitbreiding, *DataExtend* genoemd, het mogelijk om procesinstantiekenmerken de beslissingspunten in een model te laten verklaren. Als gevolg daarvan kan men op die manieer ook decision mining technieken evalueren.

Het tweede onderzoeksdoel bestaat erin de GED methodologie te incorporeren in een nieuwe evaluatieprocedure voor process discovery technieken. De nieuwe procedure focust op het meten van de kwaliteit van een techniek om het onderliggende proces te herontdekken, onafhankelijk van de gebruikte procesnotatie. De procedure vertrekt vanuit een modelpopulatie van waaruit willekeurige referentiemodellen getrokken worden. Vervolgens, meet de procedure de kwaliteit van een discovery techniek met behulp van een classificatiemethode die de kennis van het referentiemodel hanteert. Twee experimenten met vier process discovery technieken die verschillende procesnotaties hanteren hebben aangetoond dat de nieuwe procedure de doelen van empirische process discovery evaluatie ondersteunt: het vergelijken van technieken en het analyseren van de impact van procespatronen op de kwaliteit van het ontdekte model. Bovendien kunnen de resultaten van de experimenten veralgemeend worden naar de modelpopulaties. Tot slot biedt de ontworpen implementatie van de nieuwe procedure onderzoekers de mogelijkheid om hun experimenten te delen zodat ze gemakkelijk gereproduceerd kunnen worden.

Het derde onderzoeksdoel omvat het uitbreiden van de evaluatieprocedure voor process discovery technieken tot de eerste evaluatieprocedure voor decision mining technieken. Deze nieuwe procedure integreert opnieuw de GED methodologie met de uitbreiding om referentiemodellen te genereren met procesinstantiekenmerken die de beslissingpunten beïnvloeden. In een volgende stap meet de procedure de kwaliteit van de decision mining technieken om de beslissingslogica te herontdekken op basis van de event log. De kwaliteitsmeting hanteert opnieuw een classificatiemethode die de kennis van het referentiemodel met beslissingslogica uitbuit. De experimenten tonen aan dat de nieuwe procedure toelaat om decision mining technieken te vergelijken en de impact van procespatronen, zoals het determinisme van beslissingspunten, op de kwaliteit van het ontdekte model met beslissingslogica te bepalen. Bovendien kan men, door het starten vanuit de modelpopulatie, de bekomen resultaten veralgemenen naar die populatie.

In zijn geheel beoogt deze thesis om het uitvoeren van evaluatie-experimenten te stimuleren en aan te zetten tot nog meer onderzoek naar empirische evaluatie van process discovery en decision mining technieken. Eerst en vooral ondersteunen de nieuwe evaluatieprocedures het vergelijken van technieken om onderzoekers te helpen de echte waardeverhoudingen tussen de verschillende technieken te bepalen. Dit biedt een antwoord op de vraag “welke process discovery techniek presteert het beste op event logs met moeilijk te ontdekken procesgedrag?”, bijvoorbeeld gedupliceerde activiteiten. Die antwoorden helpen onderzoekers in het beoordelen van de kwaliteitsverbetering van nieuwe technieken ten opzichte van bestaande technieken. Ten tweede, ondersteunen de nieuwe evaluatieprocedures de analyse van de impact van bepaalde procespatronen, bijvoorbeeld het determinisme van de beslissingspunten, op de kwaliteit van de modellen/logica ontdekt door de geëvalueerde technieken. Zulke beoordelingen zijn van vitaal belang om te begrijpen waarom de process discovery en decision mining technieken werken in bepaalde situaties. Tot slot kan de verworven kennis evaluatie-experimenten het onderzoeksdomein bijstaan om aanbevelingen op te stellen over hoe de meest kwaliteitsvolle process discovery of decision mining techniek in de praktijk gekozen kan worden.

Table of Contents

Page

List of Tables

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In order to remain competitive in fast changing global markets, companies shifted their focus from products and services to the way these are created and delivered to the customer [37]. As a consequence, the management of business processes gained importance. Companies started to collect more and more data about these processes supported by information systems, which resulted in an explosion of process data [99].

Process mining techniques provide a way to extract hidden knowledge from these data called event logs [37, 99]. These techniques provide empirical insights and ideas for process improvement which can help businesses to achieve exceptional performance levels. The starting point of process mining is to discover a process model directly from the event data. Such a model gives an objective view on reality by visually representing the ordering between the different process activities. This discovery of a model is called control-flow process discovery [99]. Although it abstracts from other process perspectives such as time, data and resources, it is an important source of information for understanding and improving a business process.

One can further analyze the underlying business processes by comparing the discovered model with an event log using conformance checking techniques [99].

A discovered process model may not represent all the different process paths taken by cases in an event log. Manually comparing models with event log information is tedious and error-prone. Conformance checking techniques automatically highlight cases that deviate from a (discovered) process model. This can help auditors to identify cases that deviate from the expected behavior represented in a process model. Furthermore, conformance checking techniques can also highlight deviations related to other process perspectives. For example, two activities have been performed by the same person while this should have been forbidden according to the so-called 'four-eyes' principle.

Finally, process mining allows to enhance a discovered process model with information about time, resource, and data perspectives that are available in an event log. Timing information enables us to understand the performance of a business process. Also, other case data, such as costs and customer information related to that case, may provide insights in the underlying business process. For example, an essential part of a discovered control-flow model consists of the routing decisions that influence the path that the process instances follow throughout the process. A company may offer additional services for premium customers on top of the delivered product. The control-flow model does not display this information. However, event logs may contain extra information on the process instance that can be extracted to enrich a control-flow model. The task of augmenting control-flow models with rules that explain the routing decisions in terms of characteristics of the process instance, is called decision mining [28, 90]. It gives companies insights into how routing decisions in their processes are taken and allows them to verify whether this conforms with both internal as external policies.

1.1 Motivation and challenges

Over the past two decades, most attention within process mining has been paid to the development of control-flow discovery techniques, from hereon referred to as discovery techniques/algorithms. This resulted in dozens of new discovery algorithms [33, 99]. Recently, the focus of the research on process discovery has shifted. In the early days most research focused on developing techniques that could discover particular behavioral patterns that could not be detected before. Recently, new algorithms are developed to outperform existing ones in terms of the quality of the discovered control-flow model. This shift has led to an increasing importance of the research on comparing different discovery techniques [6, 33, 89] and requires a proper comparative evaluation procedure.

Empirically evaluating process discovery algorithms by comparison requires the following high-level steps: determine evaluation objective, select data sets as input for the discovery algorithms, measure the performance of discovery algorithms, and apply statistical tests to draw general conclusions. Each of these high-level steps presents challenges that remain unsolved in process mining literature [100].

The first remaining challenge relates to **determining the research objective(s)** of the empirical evaluation. Research on process discovery evaluation has identified two general objectives: benchmark state-of-the-art algorithms on process data containing various behavioral patterns [6, 33, 115], and sensitivity analysis to study the impact of discovery algorithms' parameters on discovery results [18, 86]. These objectives include the decision on which process discovery algorithms to test. However, the choice of the appropriate algorithms is not trivial as algorithms may discover control-flow models in different languages that cannot express the same behavioral patterns. This so-called "representational bias" makes it more difficult to compare algorithms that apply different languages in a fair way, i.e. to avoid comparing apples with oranges.

A second unresolved challenge is the **selection of data sets** that are used as input of the algorithms to be tested. An empirical comparison of discovery algorithms requires large amounts of appropriate data sets. The research community has set up a repository[1] for benchmark data sets, both real-life and artificial, however, the number of data sets remains rather small: 19 real-life data sets and 14 artificial data sets (checked on May 17th 2018). Alternatively, one could artificially generate unlimited amounts of process models and event logs. However, the research on artificial generators has only focused on the algorithms to generate artificial models and logs and does not provide an answer

[1] `https://data.4tu.nl/repository/collection:event_logs`

to the question which data is appropriate. As a consequence, some behavioral process patterns that present a challenge for process discovery techniques, e.g. duplicate activity labels and long-term dependencies [99], are not supported by artificial data generators.

The third remaining challenge involves the **choice of an appropriate performance measure** to compare the performances of applied algorithms. Most evaluation approaches apply performance measures that quantify the quality of the discovered model with respect to the input event log (e.g., see [33]). However, the research domain lacks an agreement on which quality measures to use. Evaluation studies have applied different measures which makes it impossible to compare their results (e.g. [33] versus [6]). Furthermore, most of the applied measures are not independent of the modelling notation and thus require a conversion of the discovered model from one notation to another. Some conversions do not preserve the behavior precisely due to “representational bias”. However, even if a conversion guarantees behaviorally similar models, it cannot always guarantee equal performance scores.

The final remaining challenge is related to the statistical tests applied during empirical evaluation. A researcher applies statistical tests to generalize the results from their experiments to make a general statement: e.g., on average algorithm A outperforms algorithm B when applied to processes with looping behavior. However, the validity of these generalizations is based on the design of the experiment, e.g. algorithms A and B should be tested on random samples of event logs with looping behavior while controlling for other process behavior. Little research in the process mining domain has focused on **designing experiments**. A correct experimental design would prevent a generalization based on a non-random sample of event logs.

Similar to control-flow process discovery, also the subdomain of decision mining has recently seen a surge in the development of new techniques [30]. The increasing number of decision mining techniques raises the importance of evaluating the quality of the returned process models with decision rules. Currently, no standard evaluation procedure has been proposed in literature. Most techniques have only been evaluated informally. Existing formal evaluations, such as [28, 74], have applied different approaches for data selection and quality measurement. Furthermore, due to their small scale, no statistical tests can be applied to determine the significance of the results. As a consequence, it is impossible to compare the techniques objectively which is necessary for the research domain to gain insights in the strengths and weaknesses of the existing approaches. Therefore the challenge remains to provide **data selection and quality measurement approaches embedded in a standard evaluation procedure that allows for the objective comparison of decision mining techniques**.

As long as the research community cannot overcome the remaining challenges of process discovery and decision mining evaluation, there will be no consensus on the quality of the available techniques. The goal of the evaluation of discovery and decision mining techniques is to understand which algorithms perform well on which process data, i.e. having different behavioral patterns that are relevant to the "real world". This thesis will tackle the data set selection, performance measurement and experimental design challenges that are currently unresolved for process discovery evaluation. Additionally, the thesis will work on the remaining challenge to provide a standard evaluation procedure for the objective comparison of decision mining techniques.

1.2 Research objective

The above defined subset of remaining challenges for process discovery and decision mining evaluation lead to the following main research objective:

Design empirical evaluation procedures for both process discovery and decision mining that enable objective comparison and generalization of results with the goal to understand why and when an algorithm works.

This procedure assumes a predefined general evaluation objective, i.e. benchmarking algorithms or performing a sensitity analysis of algorithm parameters. Then the procedure defines standard methods with tool implementations for data selection, performance measurement, and statistical tests ensuring a sound experimental design. The generated procedures would enable us to determine which discovery/decision mining algorithms perform well on which process data. This enables us to understand why an algorithm works well in a particular situation and help us to objectively determine the quality ratios between different discovery/decision mining algorithms. Consequently, such knowledge would make the choice of a suitable discovery/decision mining algorithm in practice easier. Moreover, the insights of the empirical evaluations of current algorithms could be used to guide the development of superior algorithms, i.e. measured in terms of the quality of the discovered models.

The main research objective is divided in several research goals. The first goal involves the **creation of a general methodology and algorithms with tool implementation for the generation of artificial process models and event logs.** The methodology focuses on how to generate artificial models and logs for empirically evaluating process discovery/decision mining techniques. This methodology starts from a population of process models to ensure a solid experimental design of the following evaluation experiments. This design guarantees valid statistical conclusions based on the experiment results. The algorithms for model and log generation provide implementations of the new methodology that support empirical evaluations by allowing researchers to specify model populations and generate (large) random samples of models and logs. The focus lies on the control-flow perspective which can be augmented with data attributes to include the routing decision perspective. This research goal tackles the previously identified challenges for data set selection and experimental design for both process discovery and decision mining.

The second research goal comprises **a notation independent procedure with tool support for empirical process discovery evaluation.** The new procedure combines the generation of process models and event logs of the first goal together with a classification approach to empirically evaluate and compare discovery techniques. The performance measurement uses the discovered model to classify labeled test observations as allowed or not allowed. This classification approach makes the evaluation independent of the modelling notation used by the discovery algorithms. Furthermore, it proposes standard performance measures to compare algorithm performances. As such this goal tackles the challenge of performance measurement selection for process discovery evaluation.

The third research goal covers the **the development of a standard procedure for empirical decision mining evaluation.** This procedure is an extension of the process discovery procedure to decision mining. It aims at filling the currently existing gap of a general evaluation procedure for decision mining techniques including the model and log generation of the first goal with a standard performance measure.

To motivate the choices for the above research objective and research goals, the choices are positioned within the process mining field. The evaluation procedures (including the artificial data generators) developed in this thesis are situated within the process mining research domain. Process mining aims to create artefacts, i.e. in the form of techniques and methods, that aim to solve a practical problem of general interest which positions process mining within the methodological framework of Design Science Research (DSR) [55].

An important part of DSR is the evaluation of the created artefacts. Venable et al. [119] identify six different purposes for the evaluation in DSR:

- How well a designed artefact achieves its expected environmental utility, i.e. its main purpose.
- Provide evidence that the developed artefact will be useful to solve some problem or make an improvement.
- Determine whether the created artefact improves over the current state-of-the-art.
- Beside an artefact's utility other relevant attributes should also be demonstrated, e.g. functionality, completeness, consistency, accuracy, performance, reliability and usability.
- Evaluate whether the artefact has other (undesirable) impacts, otherwise known as side effects.
- Discerning why an artefact works or not.

According to Venable et al. [119] the evaluation of an artefact always starts from artificial formative setting that evolves to a naturalistic/realistic summative setting. Formative evaluations have the purpose to improve the artefact under evaluation, while the purpose of summative evaluations is to judge to what extent the outcomes of the artefact match the expectations. The way in which the evaluation of an artefact evolves depends on the circumstances and the type of the artefact under consideration. Venable et al. [119] describe four different ways (evaluation strategies):

- Quick and simple: for simple and low risk artefacts the evaluation progresses quickly to naturalistic summative evaluations.
- Human risk & effectiveness: for artefacts with social risk and/or long run impact the evaluation emphasizes artificial formative evaluation early on, but progresses quickly to more naturalistic formative and summative evaluations.
- Technical risk & efficacy: for artefacts with technical risk and/or expensive to evaluate in a real setting. The evaluation iteratively uses artificial formative evaluation in the beginning before moving to artificial summative evaluations to rigorously determine if the benefits derived from the artefact are due to the artefact and not some other factors. It ends with more naturalistic summative evaluations.
- Purely technical artefact: if the artefact is purely technical or its use is well in the future. The evaluation is entirely artificial as naturalistic experiments are irrelevant.

This thesis focuses on the *Technical risk & efficacy* and *Purely Technical* evaluation strategies as the interest mostly lies on the technical aspects of the process mining algorithms. Furthermore, the thesis focuses on the evaluation purpose related to understanding why an artefact (process mining algorithm) works and when it works. As a result, this thesis uses artificial event data to evaluate process mining algorithms. Artificial event data enables the setup of controlled experiments which helps to understand why an algorithm performs as observed in the experiments.

The evaluation strategies in this thesis with a focus on artificial event data deviate from what is often applied in other process mining evaluation research papers. In this respect, consider the papers of De Weerdt et al. [33] and Augusto et al. [6] where the primary purpose is to compare which state-of-the-art algorithm performs best on real-life event data. Not denying the importance and relevance of those evaluation studies, their purpose differs from the purpose in this thesis. The focus on artificial event data combined with the evaluation purpose to understand why an algorithm works has received less attention. The goal of this thesis is to provide evaluation procedures for process discovery and decision mining that allows to understand why an algorithm works based on artificial data that is generated in a rigorous and structured way. Additionally, the evaluation procedures also aim to enhance the reproducibility and comparability (between different studies) of the evaluation results.

The relevance of evaluating process mining techniques based on artificial data is illustrated by the use of such data in evaluations in closely related research domains such as operations research and machine learning. In operations research and vehicle routing problems in particular, a standard set of artificial problem instances has been used to compare and test different algorithms (called heuristics), shared on websites, e.g. [77, 97]. In machine learning the widely used UCI repository [2] contains also artificial data sets such as the MONK's problems data set to test and compare a wide range of induction algorithms [78]. Moreover, researchers have developed artificial data generators for machine learning and have illustrated that the artificially generated data can be effectively used to test machine learning algorithms [81]. Finally, researchers find their way to these artificial data generators as they are incorporated in machine learning toolkits, e.g. the "make_classification" function part of the sci-kit learn package [82] for machine learning can be used to create artificial classification problems in order to test classification algorithms.

In contrast to establishing a repository of artificial data sets as done in operations research and machine learning, this thesis opts to develop an artificial data generator. The main motivation for this choice is that benchmark sets that are used for a long time may result in researchers developing algorithms that overfit the data sets in the repository [51]. Another reason for not creating a repository is that one cannot precisely estimate which effects researchers want to test in the future. As such, a generator offers the flexibility to researchers to control for the effects they intend to study.

Finally, the evaluation procedures presented in this thesis are generic rather than being fixed and exhaustive. This thesis develops the essential components of those procedures with an implementation to stimulate the uptake by other researchers. However, this thesis does not claim that the created components are exhaustive for every process discovery/decision mining evaluation experiment. Researchers can extend the procedures to accommodate it for their specific evaluation objectives, e.g. add hyperparameter optimization for the evaluated algorithms.

The following subsection describes the research methodology applied in this thesis to achieve all the research objectives.

1.3 Research methodology

The research goals presented in the previous section position this thesis within the methodological framework of design science research. “Design science is the scientific study and creation of artefacts as they are developed and used by people with the goal of solving practical problems of general interest” [55]. These artefacts can have different types: constructs, models, methods and instantiations [55]. The types of the research objectives in this thesis are methods with corresponding instantiations (implementations). Design science research requires six fundamental steps [84]: identify problem and motivate, define requirements of artefact, design and develop artefact, demonstrate artefact, evaluate artefact and communicate results. The remainder of this section will discuss how the design science steps are applied in this thesis.

This thesis starts from the observation that important challenges in the empirical evaluation of both process discovery and decision mining remain unresolved. This observation constitutes the problem identification step in design science. To get and communicate a clear and precise understanding of the problem at hand, a literature study of both process discovery, decision mining and their evaluations has been conducted and reported in the second chapter of this thesis. The challenges of data set selection, performance measurement and experimental design for process discovery evaluation and lack of a standard evaluation procedure for decision mining are illustrated and motivated based on the current state-of-the-art in process mining research.

The second step of design science research involves the description of the artefacts that could address the identified problems (challenges). This description includes the identification of specific requirements necessary for the artefact to fulfill. Section 1.2 briefly described the artefacts developed in this thesis. The second chapter will elaborate further upon these artefacts and their specific requirements. First, the requirements for the methodology on model and log generation are specified. As this artefact focuses on a valid experimental design, it will be built on principles from both statistical literature and existing process discovery/decision mining evaluation experiments. Secondly, the limitations of current artificial data generators will lead to specific requirements for the algorithms that generate process models and logs. Subsequently, literature on current process discovery evaluations reveal the requirement to decouple quality measurement from a specific modelling notation. Finally, the scarcity of formal evaluations of decision mining algorithms illustrate the lack of standard methods for data set selection and performance measurement that can be combined in a first procedure for decision mining evaluation.

Based on the description of artefacts with their detailed requirements, artefacts are designed and developed in the third step of design science research. This thesis uses ideas from experimental design in statistics to develop the methodology on model and log generation for empirical evaluation. Different from current algorithms for model and log generation, the algorithms proposed in this thesis start from the new methodology and support all the identified requirements listed in the previous step. The concepts of empirical evaluation in data mining, more specifically the classification approach, is used to design a notation independent procedure for process discovery evaluation. Subsequently, that procedure is extended to allow for decision mining evaluation.

The developed artefacts tackle the identified problems that sparked the design science research project. Therefore, the demonstration and evaluation steps of design science aim to prove that the generated artefacts can effectively solve these problems. This thesis will include a demonstration and evaluation of all generated artefacts through expert evaluation and illustrative scenarios. The artefacts are described in papers that were submitted to a peer review system of scientific conferences and journals. Moreover, the illustrative scenarios include both small- and large-scale empirical evaluations of process discovery and decision mining algorithms. Discussions of the results of these evaluations identify both strengths and weaknesses of the developed artefacts.

Finally, this thesis serves as a way to communicate the results relating to the developed artefacts. Besides the thesis, Table 1.1 lists the publications and submissions to international conferences and peer-reviewed journals to communicate the knowledge on the developed artefacts.

1.4 Outline

This thesis is organized into six chapters and its structure is illustrated in Figure 1.1. The introductory chapter presents the motivation for conducting this thesis, its research goals and research methodology. Chapter 2 then provides an overview of the process mining research domain with a focus on process discovery and decision mining. It also discusses and illustrates the challenges and limitations of empirically evaluating process discovery and decision mining. Finally, it formulates requirements for the evaluation procedures developed later in the thesis.

Chapter 3 focuses on the research goal on how to generate artificial process models and event logs for empirical process discovery and decision mining evaluation. It starts with a general methodology, followed by algorithms for control-flow data generation and ends with an extension to include data attributes that explain process routing decisions.

Chapter 4 presents a modeling notation independent procedure for empirical process discovery evaluation as discussed in the second research goal. The procedure is implemented in a tool and validated with a large process discovery evaluation experiment.

Year	Publication
2014	T. Jouck and B. Depaire, Generating Artificial Event Logs with Sufficient Discriminatory Power to Compare Process Discovery Techniques, in Proceedings of the 4th International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2014), CEUR Workshop Proceedings, 2014, pp. 174–178.
2016	T. Jouck and B. Depaire, PTandLogGenerator: a Generator for Artificial Event Data, in Proceedings of the BPM Demo Track 2016 (BPMD 2016), vol. 1789, Rio de Janeiro, 2016, CEUR workshop proceedings, pp. 23–27.
2017	J. Carmona, M. de Leoni, B. Depaire, and T. Jouck, Summary of the Process Discovery Contest 2016, in Business Process Management Workshops: BPM 2016 International Workshops Rio de Janeiro, Brazil, September 19, 2016, vol. 281 of Lecture Notes in Business Information Processing, Rio de Janeiro, 2017, Springer, pp. 7–10.
2018	T. Jouck and B. Depaire, Generating Artificial Data for Empirical Analysis of Control-flow Discovery Algorithms: A Process Tree and Log Generator, Business & Information Systems Engineering, Published online (2018).
2018	T. Jouck, A. Bolt, B. Depaire, M. de Leoni, and W.M.P. Van der Aalst, An Integrated procedure for Process Discovery Algorithm Evaluation, Business & Information Systems Engineering, Submitted, 1st review (2018).
2018	T. Jouck, M. de Leoni, and B. Depaire, Generating Decision-aware Models & Logs: Towards an Evaluation of Decision Mining, in proceedings of the 6th International Workshop on Declarative/Decision/Hybrid Mining and Modelling for Business Processes (DeHMiMoP 2018), accepted.

Table 1.1: Overview of publications in conference proceedings and scientific journals.

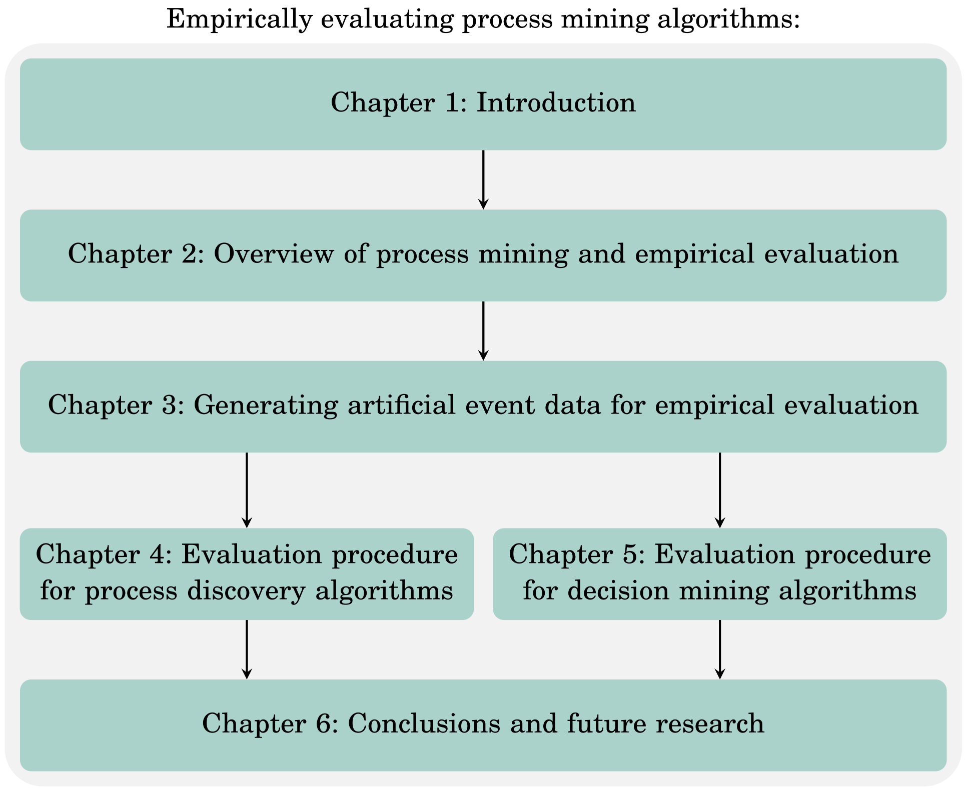

Empirically evaluating process mining algorithms:
Chapter 1: Introduction
Chapter 2: Overview of process mining and empirical evaluation
Chapter 3: Generating artificial event data for empirical evaluation
Chapter 4: Evaluation procedure for process discovery algorithms
Chapter 5: Evaluation procedure for decision mining algorithms
Chapter 6: Conclusions and future research

Figure 1.1: Outline of the thesis

Chapter 5 describes how to adapt the procedure for process discovery to enable decision mining evaluation as mentioned by research goal three. It includes an experiment to validate the procedure.

The final chapter concludes the thesis and presents future research opportunities.

CHAPTER 2

Overview of process mining, process discovery, decision mining and empirical evaluation

This chapter will provide introductory background material on process mining, process discovery and decision mining. In addition it will discuss existing empirical evaluations of both process discovery and decision mining, the remaining challenges and limitations of these evaluations and requirements of empirical evaluation procedures that aim to overcome the identified challenges.

2.1 Process mining

The focus of companies on managing and improving their business processes has led to the development of the Business Process Management (BPM) discipline. BPM is defined as "an integrated system for managing business performance by managing end-to-end business processes" [121]. BPM is a holistic management approach that includes design, implementation, monitoring, controlling and analysis of business processes [37]. As more and more business processes are supported by information systems, increasing amounts of data about the execution of these processes stored in event logs are available for analysis. This has led to the creation of the process mining discipline. Van der Aalst [99] defines process mining as “extracting knowledge from event logs in today’s systems to discover, monitor and improve real processes, i.e. not assumed processes”. It differs from pure data mining techniques as it focuses on end-to-end processes which cannot be tackled by traditional data mining techniques. Process mining supports the monitoring and analysis tasks in BPM by giving insights in the actual way of working discovered from event logs. As such event logs are the center point of process mining analyses.

2.1.1 Event log

An event log is a data set that contains information on the execution of a single business process supported by information systems. A necessary condition to extract an event log is that the information system is a Process-Aware Information Systems (PAIS), i.e. it has a notion of the end-to-end process and is not limited to support only isolated activities [99]. Examples of PAISs are: Enterprise Resource Planning systems (ERP), Customer Relationship Management systems (CRM) and Business Process Management systems (BPMS). However, most PAISs store the event data in a relational database structure where data is scattered over multiple tables on different levels of granularity. Given this, building an event log in which events are related to a single process instance on the same granularity level is challenging and requires a step-wise procedure such as proposed in [48] and [21].

An event log covers *what* has been done *when* by *whom* in relation to *which* process instance [37, 99]. Therefore, we could state that an event log contains at least the following:

- A case identifier, or process instance identifier, which relates multiple events to a specific case or process instance.

- An event label that relates an event to a well-defined step or activity instance in the process identified by an activity name.

- Ordering information on the events within a case. Ideally each event contains a timestamp, yet this is not necessary as long as there exists a total order between the events related to one case.

Additionally, an event log can contain case and event attributes on top of the case identifier and activity names. Typical examples of case and event attributes are resources, timestamps and costs. The concepts of an event, a case and an event log are formalized in the following definitions adapted from [99]:

Definition 2.1 (Event, attribute)**.** Let $\mathscr{E}$ be the event universe, i.e. the set of all possible events. Events have an activity attribute and possibly also other attributes. Let AN be a set of attribute names. Then, for any event $e \in \mathscr{E}$ and an attribute name $n \in AN$, the function $\#_n(e)$ returns the value of the attribute with name n for event e. For example, $\#_{activity}(e)$ returns the value of the mandatory event attribute activity. If event e does not have an attribute with name n then $\#_n(e) = null$.

Definition 2.2 (Case, Trace, Event Log)**.** Let $\mathscr{C}$ be the case universe, i.e. the set of all possible cases. A case has a mandatory attribute of a trace and possibly also other attributes. Any case $c \in \mathscr{C}$ and an attribute name $n \in AN : \#_n(c)$ returns the value of the attribute with name n for case c. If case c does not have an attribute with name n, then $\#_n(c) = null$. The attribute function $\#_{trace}(c)$ returns the trace $\sigma \in \mathscr{E}^*$ which is a finite sequence of events from the set of all finite sequences of events. Each event appears only once in a trace, i.e. $1 \leq i < j \leq |\sigma| : \sigma(i) \neq \sigma(j)$ with $|\sigma|$ denoting the length of the trace. An event log is a set of cases $L \subseteq \mathscr{C}$ such that each event appears only once in the entire log, i.e. for any $c_1, c_2 \in L$ such that $c_1 \neq c_2 : set(\#_{trace}(c_1)) \cap set(\#_{trace}(c_2)) = \emptyset$ with *set* a function that converts a sequence into a set.

Most process discovery techniques only require the trace information, i.e. the grouping and ordering of events, and neglect the other attributes in the event log. A log with only trace data is called a *simple event log* and is defined as follows:

Case ID	Event ID	Timestamp	Activity	Resource	Cost
1	1	2018/03/30 23:47:34	issue order	Lore	
1	2	2018/03/31 01:11:46	produce order	Jan	829
1	3	2018/03/31 03:22:54	inspect products normally	Christoph	
1	4	2018/03/31 05:47:29	package products	Tim	
2	5	2018/03/31 06:05:38	issue order	Lien	
2	6	2018/03/31 07:11:09	produce order	Tim	4224
1	7	2018/03/31 08:16:46	send invoice	Robin	
2	8	2018/03/31 14:23:54	inspect products thoroughly	Christoph	
2	9	2018/03/31 18:47:59	package products	Jan	
2	10	2018/04/01 08:16:46	deliver goods	Katrien	
1	11	2018/04/01 08:40:42	deliver goods	Katrien	
2	12	2018/04/02 10:40:42	send invoice	Robin	

Table 2.1: A fragment of an example event log in a make-to-order process

Definition 2.3 (Simple Event Log)**.** Let $A \subseteq \mathscr{A}$ be a finite set of activities. A simple trace $\sigma \in A^*$ is a sequence of activities. A log $L \in \mathbb{B}(A^*)$ is a multiset of traces. The size of the log is $|L| = t$.

Table 2.1 contains a fragment of an example event log about a make-to-order process. The fragment contains all events related to two cases. Each event has a timestamp, an activity and a resource attribute. The events related to the "produce order" activity also have a cost attribute. The simple traces of the two cases are denoted as the following sequences:
⟨ issue order, produce order, inspect products normally, package products, send invoice, deliver goods ⟩,
⟨ issue order, produce order, inspect products thoroughly, package products, deliver goods, send invoice ⟩.

2.1.2 Process models

Next to event logs, process models also play a key role in process mining analysis. Process models are used as visual representations that help people to understand, communicate about, specify and analyze operational business processes [99]. Business Process Model and Notation (BPMN) [46] and Petri nets [80] are two examples of the many existing process modeling notations.

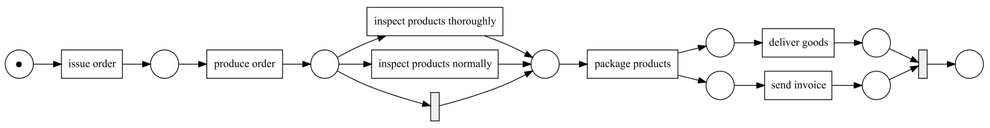


Figure 2.1: Example Petri net describing the make-to-order process.

Different notations allow users to include different process information into a process model. However, most of the notations have in common that they describe processes in terms of activities and the ordering between those activities using directed graphs.

The Petri net notation is used by many process mining algorithms because it provides formal semantics, an intuitive graphical representation, and can represent state information [99]. A Petri net is a directed graph that consists of places, transitions and flow relations (arcs). The arcs are either from a transition to a place or from a place to a transition. A place is represented with a circle and a transition with a box. Each transition represents an activity in the process, shown by a label in the transition box. A place can contain zero or more tokens. The Petri net in Figure 2.1 describes the make-to-order process introduced in the previous section.

The state of a Petri net is marked by the distribution of tokens over the places in the net. The example model contains a token in the first place in the net which indicates that no activity has been executed for that particular case. The tokens can move through the net to represent the progress of a case in a process. The flow of tokens is governed by the firing rule. If each of the input places of a transition contains a token, then the transition can fire by consuming one token from each input place and producing one token in each output place. In the example the transition “issue order” is enabled. By firing that transition the only token in the input place is consumed and a token in the only output place is produced. The firing enables the “produce order” transition. Firing the “produce order” transition enables three transitions: transition “inspect products normally”, transition “inspect products thoroughly” and a so-called silent transition (colored in grey and without label) to indicate the skipping of an inspection. Only one of those three enabled transitions can fire for a case because, if for example “inspect products thoroughly” fires, the token in the input place of each of those three transitions is consumed. Once a token reaches the last place of the net, no transition can be fired anymore which indicates that the process has ended for that particular case.

Researchers have introduced extensions to Petri nets to add extra process information to the model besides the ordering of activities. One extension that is often used by decision mining techniques is a Data Petri net. These nets include infomation on the interplay between case attributes and activities. Transitions can read and write case attributes. Moreover, a transition can have a case-attribute dependent guard that blocks the execution of an activity when it evaluates to false. Such a guard can be expressed as a rule combining different case attributes using logical operators. As such the firing rule is extended such that a transition can only fire when a transition is enabled AND the guard evaluates to true. Figure 2.4 shows a Data Petri net of the make-to-order process. The “produce order” transition writes a value to the case attribute “costs”. Transition “inspect products normally”, transition “inspect products thoroughly”, and the silent transition that is enabled after “produce order” all have associated guards depicted on the input arcs. For example, transition “inspect products thoroughly” can only fire when enabled and the “costs” attribute contains a value higher than 991.

The introductions to process models and event logs enable us to proceed to a more detailed description of process mining in the next section.

2.1.3 Types of process mining and process perspectives

There are three main types of process mining techniques [99]: process discovery, conformance checking and enhancement as illustrated by Figure 2.2. Process discovery techniques induce a (process) model from an event log without any a-priori knowledge. These techniques provide insights on how a process is actually executed and how people are working together. Conformance checking techniques compare an existing process model with an event log associated with the same process. They can be used to examine whether reality, as presented in the log, conforms with a process model and vice versa. Finally, enhancement

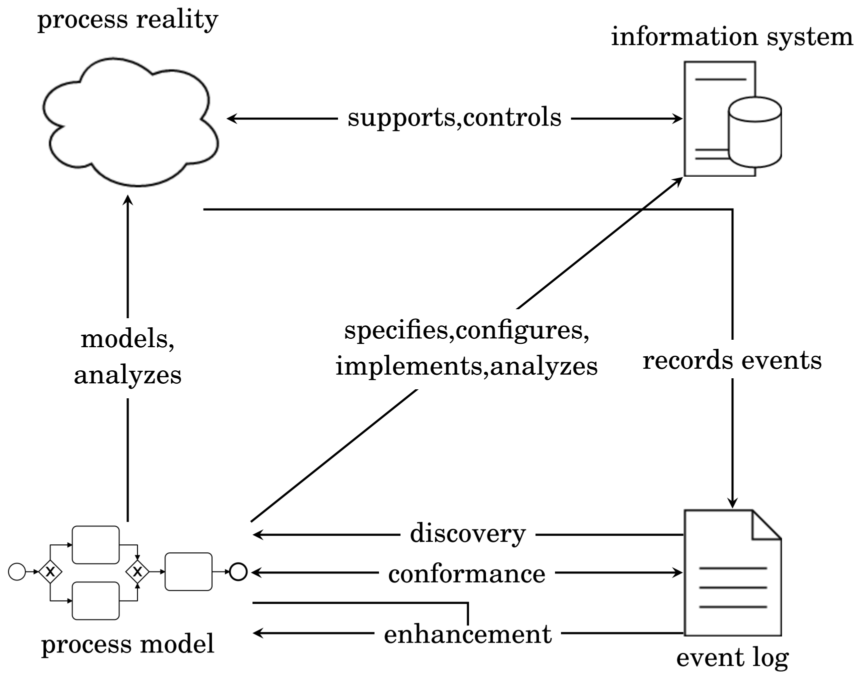


Figure 2.2: Overview of the three main types of process mining. Figure adapted from [99].

uses the event log to extend or improve an existing process model. One type of enhancement is repair which changes the process model to better reflect reality. Another type of enhancement involves adding another process perspective to the given model. An example is the extension of case data to explain the routing decisions in the process.

Besides the above types, process mining techniques can also be categorized based on the process perspective they analyze. The control-flow perspective focuses on the ordering of activities in a process. Activities can be for example sequential, concurrent or mutually exclusive. Secondly, the organizational perspective concentrates on the resources, i.e. people, systems, roles and departments, executing or consumed by the activities in a process. Furthermore, one can look into the case or data perspective of processes to study, for example, the costs of executing a case or an activity. Finally, the time perspective zooms

		Process perspective			
		Control-flow	Organizational	Case/data flow	Time
Type	Discovery	control flow process discovery	social network analysis	trace analysis	duration analysis
	Conformance	conformance checking, delta analysis	four-eyes principle	fraud detection	timed replay
	Enhancement	model repair	organizational process models	decision mining	time overlay

Table 2.2: Categorization of process mining tasks using type and process perspective as dimensions, adapted from [31].

in on, for example, the throughput times of cases, the waiting times of certain activities and discovering bottlenecks.

As a summary, Table 2.2 categorizes process mining techniques using both the type and process perspective dimensions (similar to [31]). Important to mention is that this table does not contain all possible process mining task categories, but rather gives examples for each category. This thesis focuses on control-flow process discovery and decision mining (highlighted in Table 2.2), the next subsections will discuss these categories in more detail.

2.1.4 Control-flow process discovery

Control-flow process discovery, referred to as process discovery, aims at the visualization of the process executions in an event log in a control-flow process model [31, 99]. The discovered model should be "representative" for the behavior seen in the log. However, there is no standard quality criterion of the "representative" notion. The type and correctness of the process data in the input event log also influence whether or not a discovery algorithm can produce a "representative" model. Most discovery techniques only require a *simple event log* as input, yet others require a "rich" event log to refine the discovered ordering relations.

Over the past two decades, most attention within the research domain of process mining has gone to the development of new discovery techniques. This

Traces
⟨ issue order, produce order, inspect products thoroughly, package products, send invoice, deliver goods ⟩^{8}
⟨ issue order, produce order, inspect products normally, package products, send invoice, deliver goods ⟩^{7}
⟨ issue order, produce order, package products, deliver goods, send invoice ⟩^{6}
⟨ issue order, produce order, inspect products normally, package products, deliver goods, send invoice ⟩^{5}
⟨ issue order, produce order, inspect products thoroughly, package products, deliver goods, send invoice ⟩^{2}
⟨ issue order, produce order, package products, send invoice, deliver goods ⟩^{2}

Table 2.3: Example simple event log L_s that consists of 30 cases representing 6 traces describing a make-to-order process.

resulted in a large amount of process discovery techniques: De Weerdt et al. [33] and Augusto et al. [6] identified 63 peer-reviewed, implemented and evaluated techniques from 1998 until January 2018. These techniques can be categorized according to the approach they apply [99, 116]:

- Algorithmic approaches, e.g. α miner [107], Inductive miner [69] and BPMN miner [24].
- Heuristic approaches, e.g. Flexible Heuristics miner [127], Fuzzy miner [42] and Fodina [117].
- Genetic approaches, e.g. Genetic miner [29] and Prodigen [124].
- Machine learning-based approaches, e.g. Process Miner [93] and AGNEsMiner [43].
- Region-based approaches, e.g. ILP miner [108] and HybridILPMiner [114].
- Partial and declarative approaches, e.g. Episode miner [66] and MINERful [36]

To illustrate process discovery, consider the simple event log L_s in Table 2.3 that consists of 30 cases representing 6 traces describing a make-to-order process.

Applying the Inductive miner [69] on log L_s and converting the model into a Petri net yields the control-flow model shown in Figure 2.3. The model shows the highly sequential character of the process. Each case starts with “issue order” followed by “produce order”. Then, a choice needs to be made between "inspect products normally", "inspect products thoroughly" and a silent transition that represents skipping an inspection. Finally, "package products" is executed, followed by "deliver goods" and "send invoice" that happen concurrently.

A control-flow model such as the one in Figure 2.3 does not display information on additional process perspectives such as case attributes that might be contained in a log. Decision mining techniques take a control-flow model as input and extend it with case attributes as explained in the next section.

2.1.5 Decision mining

Operational decisions influence the execution of business processes. Often these decisions relate to the choice between multiple alternative activities. Such decisions are modelled in process models as decision points. A control-flow model as constructed by a discovery technique does not give insights in which logic is applied when choosing between alternative activities, i.e. a routing decision. Such decisions can depend on resources available to execute an activity, deadlines that trigger exception handling or other data attributes related to the case such as costs. Decision mining techniques use case characteristics available in event logs to learn the logic for each routing decision in a given model [99]. Consequently, the main assumption of decision mining is that event logs contain the necessary case attributes that allow for an accurate explanation of the routing decisions.

During the past decade, several decision mining techniques have been developed, e.g. [10, 28, 30, 74, 90]. Most techniques transform the discovery of routing decision logic into a classification problem. The classes to be predicted are the alternative activities that occur after a decision point in the process model. The associated classifier is based on the observed case data attributes before the decision point is reached [74][1]. Theoretically, any classification technique can be used, however, most decision mining approaches (e.g. [28, 74, 90]) have used the decision tree learning technique C4.5 [85]. These decision mining approaches learn a decision tree for each decision point in a given model and then transform

[1]Here case data attributes refer to both attributes of the case as a whole and the attributes of the events associated with the case.

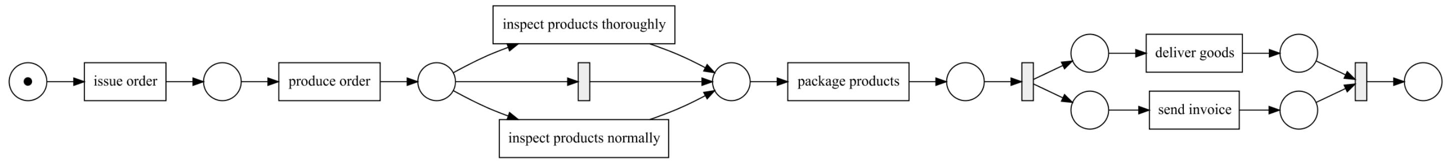


Figure 2.3: Process model discovered by the Inductive miner on the example event log L_s.

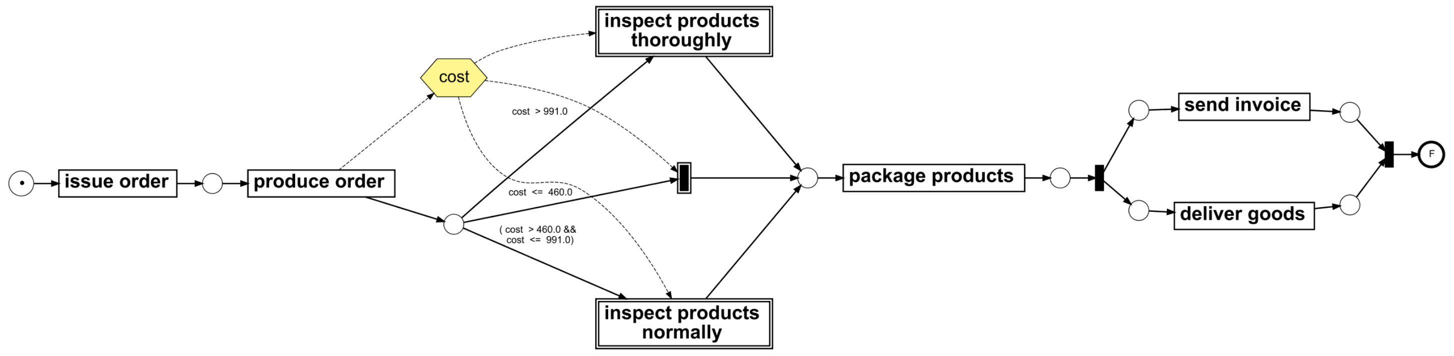


Figure 2.4: Routing logic discovered by the Data-aware decision miner on the example event log L.

these trees into decision rules. Subsequently, the models are enriched with these decision rules. For example, some transitions in a Data Petri net get a discovered rule as guard that governs the execution of that transition.

To illustrate decision mining, we have extended the simple event log in Subsection 2.1.4 with data attributes. More specifically, each event related to the activity “produce order” has an attribute “cost” denoting the cost of executing that activity. The decision mining techniques introduced in [28] uses this extra attribute to discover the routing decision logic of the choice between the activities “inspect products normally”, “inspect products thoroughly” and the silent transition (skipping the inspection). The resulting Data Petri net is displayed in Figure 2.4. The discovered rules are displayed on the branches exiting the decision point. They impose restrictions on the execution of the alternative activities:

- When the cost of “produce order” is lower than or equal to 460, the inspection is skipped.
- When the cost of “produce order” is between 460 and 991, “inspect products normally” is executed.
- When the cost of “produce order” is higher than 991, “inspect products thoroughly” is executed.

The next sections will focus on the evaluation of process discovery and decision mining techniques.

2.2 Current evaluation approaches

This section gives an overview and describes the challenges/limitations of current process discovery and decision mining approaches.

2.2.1 Process discovery evaluation approaches

Several empirical evaluation frameworks for evaluating process discovery techniques have been proposed. The seminal work of Rozinat et al. [89] introduced the first evaluation framework. That framework focuses on the comparison of discovery algorithms using conformance checking. More specifically, they apply different discovery algorithms on the same input event log and then measure the quality of the discovered models using conformance checking. The quality of a process model is characterized by four dimensions: fitness, precision, generalization, and simplicity [89, 99].

Fitness indicates how much of the behavior in the log is captured by the model. Fitness alone is not sufficient, also a proper balance between overfitting and underfitting is required [99]. A process model is overfitting (the event log) if it does not *generalize*, disallowing behavior which is part of the underlying process. This typically occurs when the model only allows for the behavior recorded in the event log. Conversely, it is underfitting (the event log) if it is not *precise*, overgeneralizing the observed behavior in the event log. Finally, simplicity refers to the preference for simpler models over more complex ones.

Rozinat et al.[89] rely on conformance checking metrics bound to the Petri net modelling notation to measure the quality of the discovered models. As such, the framework requires Petri-net based discovery algorithms or algorithms that apply modelling notations that can be converted to Petri nets. Furthermore, the Rozinat framework assumes the availability of a repository of benchmark process models and event logs. It does not discuss which models and event logs would be appropriate for such a repository or how to select them from the repository for empirical evaluation. Finally, the framework does not elaborate on the statistical tests that need to be applied to verify whether the differences in model quality between discovery algorithms are statistically significant or not.

Many researchers have taken the Rozinat et al. framework as a starting point for process discovery evaluation. Some researchers have extended the framework [87, 125] to include prediction. Others have used (an adapted version of) it to perform benchmarking studies [6, 33, 115]. Although they have made important contributions to process discovery evaluation, the gaps of the Rozinat framework related to the reliance on a benchmark repository, Petri-net-based quality measurement, and statistical tests are not completely solved.

Wang et al. [125] and Ribeiro et al. [87] have extended the Rozinat framework to evaluate and predict the best algorithm. Although the prediction can help companies in deciding which algorithm is the best to use on their event logs, the two extensions do not address how to select appropriate data sets or how to adapt quality measurement to non-Petri-net-based discovery algorithms.

De Weerdt et al. [31] and Augusto et al. [6] have reviewed and benchmarked a large set of state-of-the-art discovery algorithms on a repository of artificial and real-life event logs. These benchmarks gave important insights in the robustness of process discovery algorithms when applied in reality. The former study incremented the Rozinat framework with suitable statistical tests to compare the quality of discovery techniques based on benchmarks. These statistical tests included the combination of the fitness and precision quality measures into the F-measure. The latter study offers a benchmarking toolset that enables the empirical evaluation of a large set of state-of-the-art process discovery techniques with the latest Petri-net-based conformance checking metrics. As such, these benchmarking studies start from a repository of artificial and/or real-life logs without focusing on whether it contains the appropriate data for empirical evaluation. Furthermore, the evaluation results based on a repository of event logs cannot be generalized to process populations as the repository typically lacks reference models representing the underlying processes that produced these event logs. Additionally, they also adopt the quality measurement using Petri-net-based conformance metrics.

Finally, Vanden Broucke et al. [115] have manually created process models and event logs to study the effect of event log characteristics on process discovery results. This study has raised the importance of selecting the appropriate data sets for process discovery evaluation, yet it does not provide a formal method to tackle it.

A different approach from Rozinat et al. [89] is taken by Weber et al. [126] who adopts a probabilistic perspective. More specifically, they view processes as distributions over traces of activities and discovery algorithms as learning those distributions. The distribution of a discovered model is compared with the distribution of an artificial “ground truth” model. As such Weber et al. provide an alternative for quality measurement that is not bound to Petri nets. However, this alternative relies on the specification of probability formulae for each discovery algorithm. Up till now, these specifications are only demonstrated for the α miner and assume acyclic processes. Furthermore, Weber et al. do not specify any method for generating appropriate artificial ground truth models which are needed in their evaluation framework.

Although the above discussed empirical evaluation frameworks and studies have made important advances to the evaluation of process discovery techniques, the reliance on a benchmark repository, modelling notation dependent quality measurement, and statistical tests on real-life benchmark sets present some unresolved challenges. These challenges, in our eyes, impede the search of the process mining community to understand the true quality of any discovery technique and the quality ratios between different discovery techniques. To convince the reader of this, the next subsection will describe each of those challenges in more detail.

2.2.2 Challenges of process discovery evaluation

The discussion of challenges is structured around the general evaluation steps needed to empirically evaluate and compare learning algorithms as described by Japkowicz and Shah [51]. Any evaluation should start from some predefined evaluation objectives, e.g., benchmarking state-of-the-art algorithms on process data containing various behavioral patterns such as loops [6, 33, 115]. Benchmarking compares the quality of different discovery algorithms to *rediscover* the underlying process, given a fraction of its behavior (event log). Another example objective involves a sensitivity analysis to study the impact of discovery algorithms' parameters on discovery results [18, 86]. Additionally, the selection of process discovery algorithms to compare/evaluate is also part of the evaluation objectives.

Discovery algorithms use different target modelling languages to represent the discovered model. This target language involves implicit assumptions that limit the search space of the discovery technique: processes that cannot be represented by the target language cannot be discovered. These assumptions are known as "representational bias". Examples of discovery algorithms that apply different representational biases are the α miner [107], the Inductive miner [69] and the BPMN miner [24]. The α miner discovers Petri nets without invisible transitions and therefore cannot represent multi-choice (“or”) process behavior.[2] In contrast, the Inductive miner and BPMN miner *can* discover these multi-choice constructs. Furthermore, the BPMN miner can discover hierarchical BPMN models while the Inductive and α miner cannot discover such models. These differences in representational bias affect the selection of discovery algorithms to be compared. Although this thesis will not focus on studying and improving the “representational bias” for process discovery, it is an important criterium when selecting the discovery algorithms to compare.

Another important assumption of process discovery algorithms is the assumed completeness of the input event log. Some algorithms require more complete event logs than others, e.g. the language-based regions technique [13] assumes a global complete event log, whereas the Inductive miner [69] requires directly-follows complete event logs, which is a much weaker notion of completeness. Although algorithm selection is not the focus of this thesis, the experiments in Chapter 4 will take the completeness of the input event logs into account.

Based on the evaluation objective(s), researchers need to select the data sets on which the algorithms are tested, determine how to measure the quality of the discovered models, and, finally, apply the appropriate statistical test to draw generally valid conclusions on the evaluation results. Figure 2.5 gives an overview of the evaluation steps.

2.2.2.1 Data set selection

The first challenge faced by current evaluation approaches involves the question: how to select the data sets for evaluation? None of current approaches specifies a methodology on how to select the appropriate data sets for evaluating process discovery techniques. Most of the times, the assumption is made that the current repository of real-life event logs[3] should be used as benchmarks. This presents two important limitations to empirical process discovery evaluation. First of all, the process population characteristics of a real event log are unknown because a

[2]For example, if activity a and b are in multi-choice, then either one of the two activities is performed, either both activities can be performed in any order.

[3]`https://data.4tu.nl/repository/collection:event_logs`

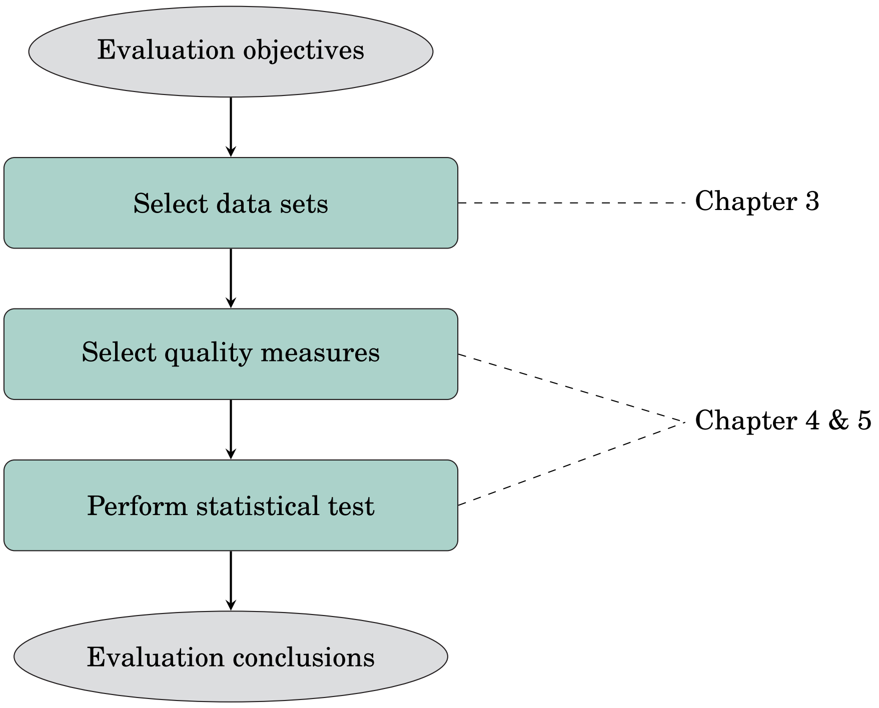


Figure 2.5: General evaluation steps adapted from [51]. The chapters that focus on specific steps are indicated with dashed lines.

reference model is lacking. Therefore, the types of process behavior are in the event log are unknown. Nevertheless, this is typically required when testing if an algorithm can handle certain process behavior, e.g. loops. Secondly, the number of real event logs in the repository is limited, i.e. 19 publicly shared data sets. However, in order to draw statistically significant conclusions, one needs large amounts of data sets.

As an alternative, researchers have proposed to artificially generate event data to overcome both of these limitations [100]. Several approaches exist for generating artificial process models and event logs [17, 54, 63, 112]. Each of these approaches focus on the algorithms and implementation of generating artificial models and event logs. However, none of the existing approaches presents a general methodology of how to generate event data for empirically evaluating process discovery techniques. Such a methodology, nonetheless, is an essential starting point to ensure that the empirical analysis of artificial data follows a sound experimental design that guarantees valid statistical claims (discussed in more detail in Section 2.2.2.3). Furthermore, as the current artificial generators do not start from an evaluation methodology but rather focus on the algorithms to generate the data, the existing generators do not provide an answer to the question which data is appropriate for process discovery evaluation. As a result, several limitations exist on the process behavior included in the generated models and logs. For example, long-term dependencies and reoccurring acitivities[4] present challenges for process discovery algorithms [99], but are not supported by current artificial generators. This illustrates the need for enhanced artificial model generators.

2.2.2.2 Measuring quality of discovery results

The second challenge involves the measurement of the algorithms performance, here the quality of the discovered models. Most evaluation approaches, i.e. [6, 33, 86, 87, 89, 115, 117, 125, 126], have focused on the use of conformance checking to measure the quality of the discovery results. The applied conformance checking metrics are bound to the Petri net modelling notation. As a result, the selection of discovery algorithms is restricted to Petri net-based discovery algorithms or algorithms that apply modelling notations that can be converted to Petri nets. However, the conversion may result in unfair comparisons, illustrated by the following example.

Consider an event log (see log L_{or} in Appendix A) similar to the make-to-order process in Section 2.1.4 except that the exclusive choice between “inspect products normally”, “inspect products thoroughly” and skipping the inspection is changed to a multi-choice (OR) between “inspect products normally” and “inspect products thoroughly”. Then we apply the Inductive miner (IM) [69] and the Flexible heuristics miner (FHM) [127]. The IM uses process trees as modelling notation. Process trees are block-structured models that can be decomposed in properly nested subprocesses such that each subprocess has single entry

[4]The same activity that appears in different parts of the process, indicating that the activity can reoccur. Reoccurring activities are typically modelled using duplicate activity labels, hence the term “duplicate activities” is often used in literature.

and exit points [69]. Block-struturedness guarantees soundness. Soundness is a correctness criterion such that a sound process model is free of deadlocks, livelocks and other anomalies [99]. The FHM on the other hand, uses causal nets as modelling language. These types of directed graphs are specifically tailored towards process discovery as they only represent causal dependencies between activities in a process. They can also expresss non-block-structured processes, i.e. they do not guarantee soundness. Both process trees and causal nets can be converted to Petri nets [99, 101] to calculate their precision using the one-align precision metric [3].

These conversions result in the models in Figure 2.6 and Figure 2.7. Although both algorithms correctly rediscovered the multi-choice, the conversion of the causal net into a Petri net introduced an error on the join semantics before "package products". This error may cause the activities "package products", "deliver goods" and "send invoice" to happen twice instead of once. This conversion error is a result from the fact that Petri nets are not as expressive as causal nets. Therefore, the converted Petri net may over-approximate the behavior in the discovered causal net, i.e. the net may allow for more behavior than allowed by the discovered causal net [116]. The conversion error results in a different precision value for the two converted Petri nets while they should have been exactly the same based on the behavior of the discovered models.

Moreover, even without conversion errors, two behaviorally equivalent models may produce different precision scores. Consider the same event log with multi-choice behavior as used above. If we apply the trace miner that lists each possible trace in the log as a separate sequential fragment in a Petri net, we obtain the "enumerating" model in Figure 2.8. The model discovered by the IM (Figure 2.6) contains exactly the same process behavior as the model discovered by the trace miner, no more and no less, and therefore should have the same precision score. However, the one-align precision metric [3] computes a precison of 0.9068 for the model discovered by the Inductive miner and 1 for the "enumerating" model. The observation that the one-align precision metric computes a different precision score for two models with exactly the same behavior was also shown by Tax et al. [96]. It should be noted that more recent global precision metrics, such as the anti-alignments precision metric [109], have alleviated this

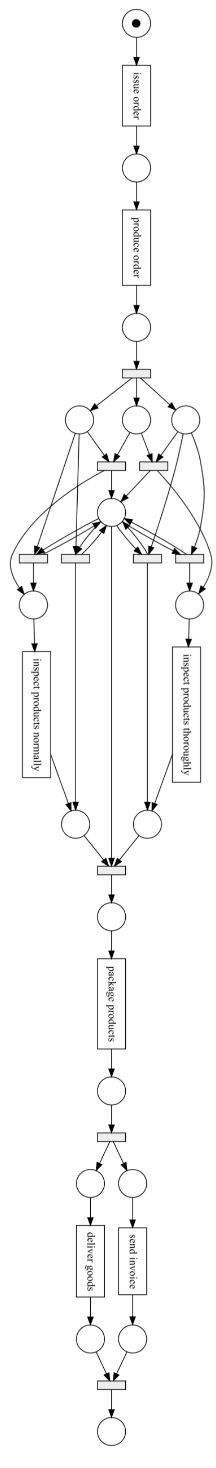


Figure 2.6: Converted Petri net discovered by the Inductive miner on event log with OR.

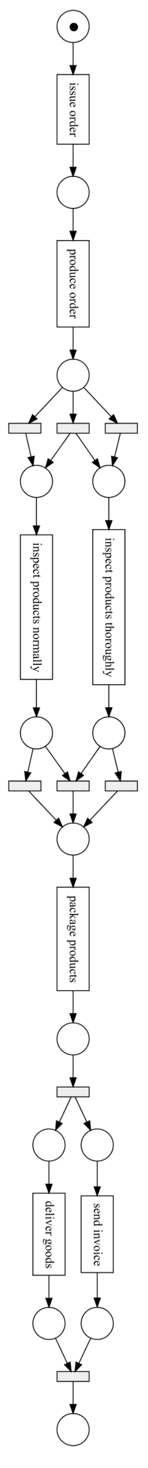


Figure 2.7: Converted Petri net discovered by the Flexible heuristics miner on event log with OR.

problem and compute equal precision scores for the “enumerating” model and the model discovered by the IM.

The above examples illustrated that differences in precision due to conversion and model representation are only artificial. Although all three algorithms could discover the same behavior and therefore do not differ in the precision quality dimension, the precision metric assigned different scores to the (converted) discovered models. The differences in representational bias are tackled by more recent conformance checking metrics. However, the conversion is the main problem as it would still lead to different precision scores for behaviorally equivalent models that are in different modeling notations. As such we state that conformance checking metrics bound to a particular modeling notation may favor an algorithm that uses the same notation over another that uses a different notation. However, we argue that this “better fit” should not result in a different score for a quality metric as this would result in an unfair comparison between discovery algorithms.

2.2.2.3 Statistical tests

The final challenge relates to the statistical tests in discovery evaluation. Once the discovery algorithms are applied and the quality of the discovered models are measured, a statistical analysis is needed to draw general conclusions on the evaluation results. For example, when multiple discovery algorithms are benchmarked on a set of real-life logs, the statistical test can verify whether the differences in performances are statistically significant or not. In case the differences are statistically significant, one can state that on average a technique outperforms some other techniques not only on that particular set of real-life event logs, but also generalized to the process population from which that set originates. For example, De Weerdt et al. [33] found that the Heuristics miner [128] significantly outperforms the α miner among others on their benchmark set of real-life event logs. Moreover, they conclude that Heuristics miner seems the most appropriate and robust in a real-life context in terms of scalability and fitness, precision, and simplicity of the discovered model.

Evaluations such as [6, 33] on real-life event logs are useful to verify whether

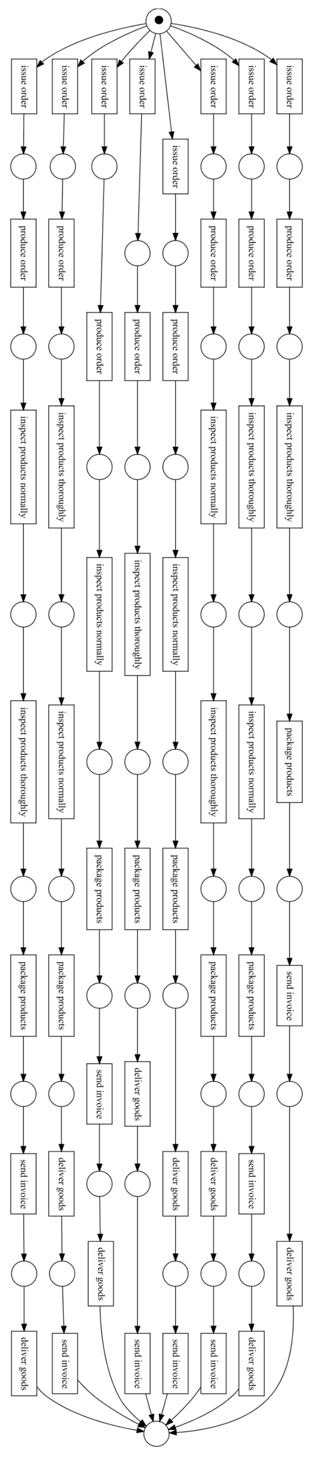


Figure 2.8: Petri net discovered by the trace miner on event log with OR.

we can actually use certain discovery techniques in a practical setting. However, we lack a reference model and therefore do not know from which process population these event logs are samples of. In order to understand why an algorithm works, researchers need to test those algorithms in a controlled environment first and generalize to a model population. For example, because existing discovery algorithms have problems to rediscover duplicate activities, a new discovery algorithm specifically focused on discovering duplicate activities is developed. In order to test whether the new discovery algorithm can effectively achieve this, both existing algorithms and the newly developed algorithm need to be applied on event data with duplicate activities. Hence, it is important that a researcher can control for those duplicate activities as to precisely study their effect on an algorithm's ability to rediscover those duplicate activities. Additionally, a researcher would then like to generalize his/her findings to actually show that in general, i.e. in different model populations with duplicate activities, the newly developed algorithm performs better.

The relevance of the generalization to a model population is further supported by the no free lunch theorem by Wolpert and Macready [131] that establishes the following: "for any algorithm, any elevated performance over one class of problems is offset by performance over another class". Hence, it is useful to generalize for which model populations the newly developed technique performs better than existing algorithms and for which model populations it performs worse than existing algorithms as there is always a tradeoff.

Furthermore, the lack of a reference model that represents the system, i.e. the real underlying process, also impacts the performance measurement during benchmarking. The quality of an algorithm to rediscover the system can only be measured using conformance checking that compares a given log with the discovered model. This comparison results in a so-called log measure that represents the quality of discovered model with respect to the event log it was learned from. Such measures do not take into account that the event log is a limited sample of the system and possibly contains measurement errors, also known as noise. Janssenswillen et al. [50] found that the log measures are biased estimators of the similarity between the discovered model and the system in case the log is incomplete or contains noise. Therefore, the lack of a reference model impedes unbiased measurement of the discovery algorithms quality to rediscover the system. Finally, we cannot claim that the samples of real-life logs are random samples. As a consequence, the results of those evaluations cannot be generalized to process populations.

Alternatively, evaluations have used artificial process models and event logs for comparing discovery techniques [33, 86, 87, 89, 115, 125, 126]. Despite the fact that we know the process populations for those models and event logs, the samples were small and not randomly generated. Hence, the statistical validity of general claims is limited. Furthermore, the artificial process models are not generated by controlling the probability of certain process behavior to be present. As a result, the event logs generated from these models do not allow one to evaluate the correlation between the quality of the discovered models and the presence of certain process behavior. This is because the process behavior is an endogeneous variable due to the fact that it is observational in nature and not determined randomly in the experimental design [5].

Based on the above challenges, we argue that more attention should be paid to the design of the evaluation experiments to allow for valid generalizations. The experiment design relies heavily on the data set selection and the quality measurement. If we can collect random and large samples from appropriate process populations *and* we use an unbiased estimate of an algorithm's quality to rediscover the system, then statistical tests will allow us to make generally valid claims on the evaluation results.

2.2.3 Decision mining evaluation approaches and their challenges

Currently, no decision mining evaluation procedure has been proposed in literature. Therefore, we discuss the evaluation approaches taken in the papers that introduce new decision mining techniques. The papers of [10, 30, 90] have mainly focused on the development and demonstration of the proposed technique. These papers only include an informal evaluation in which the output of a technique is compared with the manually created example model or, in case of real event logs, the output is described but the quality is not measured. However, the papers [25, 28, 74] do include a formal evaluation: the evaluations of [25, 28] compared the discovered model augmented with decision rules with the artificial reference model and decision rules, the evaluations of [28, 74] calculated the accuracy of the discovered model and rules with regard to the real-life logs.

When we discuss the above evaluation approaches with the general evaluation steps for empirical evaluation and comparison as illustrated in Figure 2.5, we can conclude the following.

Firstly, the most important remaining challenge of decision mining evaluation is the data set selection for empirical evaluation. Similar to process discovery, there exists no guideline on how to select the appropriate data for decision mining evaluation. Also here the event logs from the publicly available repository[5] have been used as benchmarks. In contrast to process discovery, only 5 different event logs from the repository were used in a decision mining evaluation [28, 71] or demonstration [30]. It is unclear whether the other event logs contain case attributes that are sufficient to explain the process routing decisions as required for decision mining evaluation. Therefore, analogous to process discovery, two important limitations for empirical decision mining evaluation are the unknown process population of the real-life event logs and the limited amount of event logs in the publicly available repository.

Comparable to process discovery, researchers have included small samples of manually created artificial datasets. However, empirical evaluation of decision mining algorithms needs large amounts of data to make statistically valid conclusions. Only one artificial data generating approach supports the generation of both control-flow and case/data-flow perspectives [17]. PLG2 [17] allows for extending control-flow models with data attributes, but in a more general sense. It can add case attributes to activities such that an activity can either generate a case attribute or require a case attribute. The latter is implemented by automatically generating the required case attribute before the execution of that activity. The random model generator cannot guarantee that this case attribute requirement happens to activities after a routing decision.[6] Nevertheless, this

[5]`https://data.4tu.nl/repository/collection:event_logs_real`

[6]PLG2 allows users to add the requirements also manually, however, that would not lead to random samples and thus obstruct the generalization of evaluation results.

is necessary for the evaluation of decision mining techniques as they focus on discovering the routing decision logic. As a result, there is need of an artificial model and log generator tailored to the evaluation of decision mining techniques.

Secondly, different metrics are applied to measure performances, i.e. the quality of the discovered decision logic. The evaluations of [28, 74] use data conformance checking to compare the log to the discovered model, while [25] checks the quality with regard to the reference model. However, similar to the discussion on statistical tests for process discovery evaluation (see Section 2.2.2.3), such log measures could be biased estimators of decision miner's quality to rediscover the routing logic of the underlying process. Therefore, the lack of a reference model impedes the unbiased quality measurement of decision mining techniques.

Finally, due to the lack of large datasets and extensive evaluations, the statistical test selection is often neglected and replaced by simply comparing absolute figures of quality measures.

The challenges of both process discovery and decision mining evaluation related to data set selection, quality measurement and statistical tests lead us to the specify the requirements of the artefacts to overcome these challenges in the next section.

2.3 Requirements of artefacts

The research goals in Section 1.2 describe the artefacts generated in this thesis to overcome the challenges discussed in the previous section. Because we apply the principles of Design Science research, we will elaborate on the specific requirements for each of these artefacts to specify how they will overcome the stated challenges.

2.3.1 Requirements for artificial model and log generation

Chapter 3 reports on the first artefact called GED (Generating articial Event Data) which includes a general methodology and algorithms for generating artificial process models and event logs for process discovery and decision mining evaluation. GED tackles the data set selection and statistical tests challenges for both process discovery and decision mining (see Sections 2.2.2.1, 2.2.2.3 and Section 2.2.3). Therefore, it should allow users to specify a model population and than draw random samples of process models and event logs from it. At the same time GED should guarantee a correct experimental design such that it enables statistically valid generalizations of evaluation results. We now describe three use cases that help us to derive more detailed requirements for GED.

One possible use case for GED is the performance comparison of discovery algorithms in terms of model quality. In this case a researcher needs to define populations with an extended set of control-flow patterns. If only a limited set of basic patterns were available, the simplicity of the event data could bias the comparison results. A second use case for GED is the goal to understand the effect of specific control-flow behavior on algorithm performance. This use case requires full control over all possible control-flow patterns in the generated models to enable causal analysis. A third use case evaluates whether decision mining algorithms can rediscover non-deterministic routing decision logic based on case attributes in the event log. A difference between decision mining algorithms is whether they assume the routing decisions to be fully deterministic [28] or not. [74]. In a fully deterministic routing decision there is only one outgoing branch possible for any combination of case attribute values. However, in reality, decisions can be non-deterministic due to conflicting or ambiguous business rules [88]. This use case requires control over the determinism of routing decisions in the generated models and logs.

2.3.1.1 Deriving detailed requirements

The leftmost column of Table 2.4 lists all requirements for the methodology. The first group of requirements regards the *full control* over the control-flow and data-flow behavior in the generated process models (control-flow patterns and routing decisions) and event logs (log characteristics). Multiple evaluation studies [29, 110, 115] have assessed discovery algorithms using an extensive set of control-flow patterns. This set includes the basic workflow control-flow patterns (WCP) [92]: sequence (WCP-1), parallelism (WCP-2 and 3), exclusive choice (WCP-4 and 5), “or” (WCP-6 and 7) and structured loops (WCP-21). The studies of Kunze et al. [65] and Zur Muehlen and Recker [132] who analyzed real-life BPMN models confirmed that those basic patterns are used in practice by consultants and trained process modelers. Besides the basic patterns, the set also covers the complex constructs invisible (skipping) activities, reoccurring activities and long-term dependencies. The mining of reoccurring activities has received more and more attention within the process mining domain in the past years, e.g. [70, 83, 123]. Van der Aalst [99] and Buijs [16] describe long-term dependencies as a key aspect of process behavior in reality. Two evaluation studies [29, 115] also investigated the effect of log characteristics, i.e. number of traces, noise and infrequent behavior.[7] The evaluations in [71, 74] assessed decision mining algorithms on non-deterministic routing decisions. Consequently, GED should support full control over all these patterns and characteristics.

Additionally, the soundness was added as a requirement for each generated model. This ensures that the produced model can never cause a livelock or a deadlock during the simulation. A simulator allowing for unsound models requires the detection of the violation and the repair of that violation which is far from trivial [16]. Simply restarting the simulation everytime a livelock or deadlocks occurs does not guarantee random samples which is another requirement.

The second group of requirements relates to *randomness*. In order to generalize findings from event logs to a process population, the event logs should be random samples to avoid biased conclusions. To be more specific, GED should allow to draw a random sample of models from a process population. Additionally, GED should support the simulation of a random sample of logs from the sample of models.

The third and last group of requirements specifies the *formats* of the generated event data and *integration* with process mining tools. A discovery evaluation experiment can exploit the extensive set of algorithms and conformance checking techniques in the ProM framework [120]. This framework uses the

[7]Noise is defined in this thesis as incorrect behavior in the log (see Section 3.3).

XES standard [1] for event logs and supports different XML-based formats for process models. Therefore, it is important that GED produces models and logs in these standard formats. An additional advantage would be the integration within the ProM framework [120] to enable automated experiments.

Each of the requirements are listed in the leftmost column of Table 2.4, grouped by category: full control, randomness, and standard formats. The use of the general methodology for artificial event data needs model and log generation algorithms plus tool implementation that support all these requirements.

2.3.1.2 Evaluation of related work

We have evaluated the existing implementations for generating event data against the requirements stated above. The results in Table 2.4 show that none of the existing tools fulfills all the requirements.

PLG2 [17, 19] is the most mature tool, but is limited to block-structured process models and therefore cannot contain LT dependencies. To illustrate this, consider the Petri net in Figure 2.9. The net without the grey places and their incoming and outgoing arcs is block-structured. The net can be decomposed in properly nested subprocesses such that each each subprocess has single entry and exit points. The first block is composed of the source place, transitions “a” and “b”, their shared output place (the white colored place), and the arcs connecting these elements. Similar to that is the block including transitions “d” and “e”, their shared input and output place and arcs. Another block is the transition “c” with its input and output places and arcs. The final block covers all the previously mentioned blocks. If we want to model that transition “e” can only fire after transition “a” and transition “d” can only fire after transition “b”, then we need to introduce the grey places and their input and output arcs.[8] These dependencies are an example of LT dependencies and they violate the block-structuredness of the net: e.g., the block including transition “a”, transition “b” would have more than one exit point. Such LT dependencies present a challenge for process discovery algorithms [99] and therefore makes the limitation of an artificial data generator to block-structured models too strict. Additionally,

[8]This is a valid solution if we do not want to have two transitions with label “c”.

R1 Full Control	PLG2 [17]	GraphGrammar [63]	BeehiveZ [54]	TestBed [112]
Number of activities		✓	✓	
Sequence (WCP-1)	✓	✓		✓
Parallel (WCP-2-3)	✓	✓		✓
Choice (WCP-4-5)	✓	✓		✓
Loop (WCP-21)	✓			✓
Or (WCP-6-7)				
Silent (skipping) activities	✓			
Reoccurring (duplicate) act.				
Long-term (LT) dependencies				✓[2]
Infrequent paths	✓			
Routing decision determinism				
Soundness	✓	✓		[2]
No. traces	✓	✓	✓	
Noise	✓		✓	
R2 Randomness				
Random Generation	✓	✓	✓	✓
R3 Standard Formats				
Models	✓		✓	✓
Logs	✓		✓	
ProM integration	✓[1]			✓

[1] PLG2 [17] is only available as a standalone tool, but the older PLG [20] is implemented in ProM with all the indicated requirements

[2] The approach allows to add “arc bridges” to create non-free choice constructs, yet it does not guarantee sound models

Table 2.4: Evaluating existing artificial generators on the requirements of GED

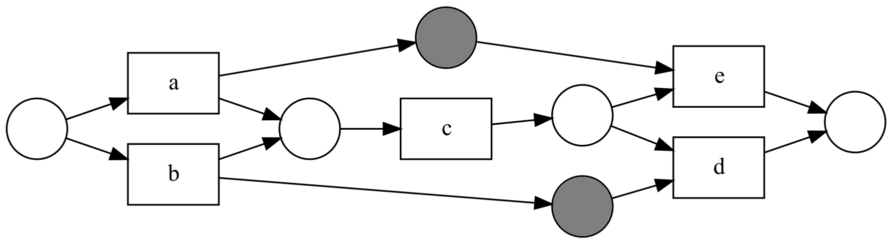


Figure 2.9: An example of non-block-structured behavior.

PLG2 is the only generator that allows for adding the data-flow perspective to control-flow models by adding case attribute requirements to any activity in the model. Nevertheless, PLG2 does not allow the user to control that these requirements happen to activities after a routing decision which is required for decision mining evaluation (cf. discussion in Section 2.2.3).

The approach using GraphGrammar [63] only allows for generating and simulating rather simplistic process models. The BeehiveZ tool [54] gives users limited control over the control-flow constructs in the generated models as users can only pick a class of process models from which a random sample is drawn and simulated. Finally, the TestBed tool [112] does not include a simulator, but allows for LT dependencies. However, to model LT dependencies it uses non-free choice constructs which do not guarantee soundness of the produced models.

An alternative solution to guarantee soundness that extends the TestBed approach would be as follows: first, generate a subclass of Petri nets (called Jackson nets [111–113]) that are always sound, then extend these models with non-free choice constructs (NFC) to introduce LT dependencies. Each time a NFC is added, check for soundness, if there is a violation, revert the NFC and try another. However, deciding soundness may be intractable for complex nets [98] and therefore this solution is insufficient. Moreover, the randomness of the generated models, another requirement of GED, is possibly violated as some random sound models are excluded if they require a series of bridge rules which first make the model unsound and later on make the model sound again resulting in a truncated sample.[9]

[9]The observations in a truncated sample are limited as the models that require bridge rules

Other approaches for artificial event data generation such as SecSy [95], CPN Tools [53] and GENA log generator [79] only focus on simulation of an event log given a process model as input. These approaches are only relevant for the log generation as discussed in Section 3.3.

As none of the current tools meets all the requirements, Chapter 3 introduces a methodology and implementation that conforms to all the requirements. We will mostly focus on the challenge of solving the trade-off between including LT dependencies while ensuring soundness of the generated models, and the extension of models and logs with data-flow for decision mining.

2.3.2 Requirements for process discovery evaluation procedure

Chapter 4 describes the second artefact which is a procedure with tool support for empirical evaluation of process discovery algorithms. This artefact will tackle the challenges of quality measurement and statistical tests (see Sections 2.2.2.2 and 2.2.2.3). The procedure starts from a defined evaluation objective such as benchmarking or sensitivity analysis, and should fullfill three major requirements: notation indepent quality measurement, solid experimental design, and support for automating empirical evaluation experiments. The following subsections will highlight the importance and details of each of these major requirements which are summarized in Table 2.5.

2.3.2.1 Notation independent quality measurement

The quality of discovered models is measured in order to compare different discovery algorithms. The *quality measurement should be independent of the modelling notation in which the discovered models are represented* to avoid the problems due to conversion and representational bias discussed in Section 2.2.2.2. The term quality measurement is broad as the quality of models can be measured along four different dimensions: fitness, precision, generalization, and simplicity (see Section 2.2.1). This thesis will focus on the fitness

that make the model first unsound before they can be made sound again are omitted from the sample.

and precision/generalization dimensions to study the efficiency of process discovery techniques to rediscover the original system and while balancing between overfitting and underfitting.

To make the quality measurement independent of the modelling notation, we focus on the behavior of a model rather than the structure of a model. The model structure is typically used for measuring simplicity and is strongly connected to the characteristics of a modelling notation. For example, the size of a BPMN model will naturally be smaller than a Petri net as BPMN abstracts from representing the state of a process instance. Therefore, we argue that further research on comparing simplicity of models in different notations is needed. As such, simplicity is left out of the quality measurement of this thesis.

Furthermore, the quality measurement should enable *unbiased estimates of an algorithm's quality to rediscover the underlying system*. Therefore, the process discovery evaluation procedure will use the data generation methodology and algorithms of the GED artefact. As such, a reference model is available that represents the behavior of the underlying process to be rediscovered by the compared discovery algorithms. This reference model should be used instead of classical log measures to get an unbiased estimate of the quality of the discovered model.

2.3.2.2 Solid experimental design

An evaluation analysis aims to test statistical hypotheses about a discovery algorithm. For example, the hypothesis: "the presence of loops causes the Alpha+ miner [106] to discover models with lower fitness". Another example of a hypothesis: "the Alpha+ miner and Heuristics miner [128] perform equally in the fitness dimension on event logs with multi-choice (OR) behavior". This makes it fit within the experimental design methodology in which the primary goal is to establish a causal connection between the independent (algorithm, log characteristics) and dependent (model quality criteria) variables [64]. The three cornerstones of good experimental design are: *randomization, replication and blocking* [40]. The three cornerstones together are fundamental requirements of our evaluation procedure (artefact) as they make the experiments scientifically sound (e.g., avoid bias or wrong conclusions).

Randomization involves the random assignment of subjects to the treatment in order to limit bias in the outcome of the experiment [64, 122]. In the evaluation context, the subjects are the event logs and the treatments are the discovery algorithms. Ultimately, the aim is to generalize the evaluation results to the process population from which the event logs originate. Therefore, the data generation step adopts three levels (as briefly discussed in Section 2.3.1.1): a process population containing all processes with the desired control-flow characteristics, a process model which is a random sample of the process population, and an event log which is a random sample from the model, i.e. a random sample drawn from the distribution of traces defined by the model. Only by adopting this hierarchical structure of random samples, one can generalize the results based on the event logs to the process population.

Replication means that more than one experimental unit is observed under the same conditions. It enables researchers to estimate error effects and obtain a more precise estimate of treatment effects [64]. In the context of process discovery this implies that one needs to test a specific algorithm on more than one event log to accurately assess the effect of that algorithm on model quality. The procedure requires that the evaluation is based on a sample of event logs from a model that is drawn from a given population to obtain better estimates of the studied effect.

Finally, blocking an experiment is dividing the observations into similar groups. In this way one can compare the variation between groups more precisely [122]. For example, if the experiment studies the effect of loops on model quality, also other characteristics such as infrequent behavior could have an effect. Therefore, the evaluation procedure should allow to vary the presence of loops in models (variable of interest) while holding the infrequent behavior constant to obtain precise estimates of the effect of loops on model quality (studied effect). To illustrate this, consider an example where both the presence of loops and infrequent behavior are varied at the same time. If the statistical analysis considers loops, but omits infrequent paths as an explanation of model quality, the learned effect of loops on model quality could be biased. Such bias happens if infrequent paths have an effect on model quality which is now incorrectly

R1 Notation independence
Notation independent fitness measurement
Notation independent precision/generalization measurement
Unbiased estimates using reference model knowledge
R2 Experimental design
Randomization
Replication
Blocking
R3 Automation
Connect and automate evaluation steps
Allow shareable evaluation setup

Table 2.5: Requirements of process discovery evaluation procedure

in the learned effect of loops on model quality. Here blocking, i.e. holding the infrequent paths constant, would be needed if one wants to learn the true effect of loops (and only loops) on model quality.

2.3.2.3 Automating evaluation experiments

Experiments evaluating discovery techniques involve large-scale and computationally expensive experiments that require intensive human assistance [14]. Therefore, automating these experiments removes the need for the human assistance and reduces the time needed to perform experiments. This results in the requirement of *an evaluation tool that connects the fundamental steps*, i.e. data set selection, quality measurement, and statistical tests, to enable large-scale evaluation experiments. Furthermore, this tool should allow to *share evaluation setups*, i.e. all parameters related to data selection, discovery algorithms, quality measurement, and statistical tests, to ensure that experiments can be reproduced and extended by other researchers.

2.3.3 Requirements for decision mining evaluation procedure

Chapter 5 will present the third artefact that involves a first evaluation procedure for decision mining techniques. This procedure should combine the *data* *selection step* provided by the GED artefact (see Section 2.3.1) and extend the *quality measurement using reference model knowledge* of the second artefact (see Section 2.3.2) to assess the quality of the discovered routing decision logic.

2.4 Conclusion

This chapter has provided the reader with an introduction to process mining, process discovery and decision mining. The latter two have been positioned within the process mining field and explained and illustrated with a small example. The second part of the chapter has focused on empirically evaluating process discovery and decision mining techniques. A critical view on the methodological aspect of current evaluation approaches has resulted in a list of challenges that currently prevent objective comparison and generalization of results. Finally, each of the identified challenges has led to a set of requirements for improved empirical evaluation approaches for both process discovery and decision mining techniques which constitute the main research objective of this thesis.

CHAPTER 3

GENERATING ARTIFICIAL EVENT DATA FOR PROCESS DISCOVERY AND DECISION MINING EVALUATION

This chapter will describe the Generating artificial Event Data (GED) artefact for empirical evaluation of both process discovery and decision mining techniques. GED tackles the challenges related to data set selection and statistical tests as described in Sections 2.2.2.1 and 2.2.2.3. In order to do this, GED will include the full control, randomness and standard formats requirements stated in Section 2.3.1. The GED artefact includes a methodology and implementation called the "Process Tree and Log Generator" (PTandLogGenerator) to generate artificial process models and event logs for process discovery evaluation. Secondly, it introduces an extension to the "PTandLogGenerator" called "DataExtend" that generates event logs with case attributes for decision mining evaluation. More specifically, GED makes the following contributions:

- A general methodology for the generation of random artificial control-flow process models and event logs that uses a hierarchical experimental design (Section 3.1).
- An algorithm for generating random control-flow process models from a

predefined population of processes. The algorithm guarantees soundness of the generated models and allows for a specific type of non-block-structured behavior, namely long-term dependencies (Section 3.2).

- An algorithm for simulating the generated models into a random sample of event logs (Section 3.3).
- An extension to the model and log generation algorithms to include case attributes that represent the routing decision logic (Section 3.4).

The material in this chapter is based on the work published in [58, 60, 61]. The chapter is structured according to the above contributions: it starts with the general methodology, then it discusses both model and log generation algorithms, followed by the data extension. The demonstration and evaluation will illustrate and assess the presented algorithms, followed by a discussion on the limitations and threats to validity, and, finally, a conclusion summarizes the chapter.

3.1 Generating artificial Event Data methodology

The starting point of the chapter is a new methodology called the GED methodology for generating artificial models and logs for *process discovery evaluation* (and extentable to decision mining evaluation see Section 3.4). This methodology (illustrated in Figure 3.1) consolidates concepts of experimental design in statistics with existing process mining research methodology. To our knowledge, this is the first time that a methodology combines the ideas of those two research areas. The GED methodology is the blueprint for our model and log generation algorithms later in this chapter.

This section focuses on the control-flow perspective of the generated models and logs. However, the methodology can be adapted to include also case attributes for decision mining evaluation which is discussed in Section 3.4.

The GED methodology uses a hierarchical experimental design [15] for the generated event data. Figure 3.2 illustrates the design: the first level comprises the process model population (hereafter called model population), the second level a random sample of process models and the third level a random sample of

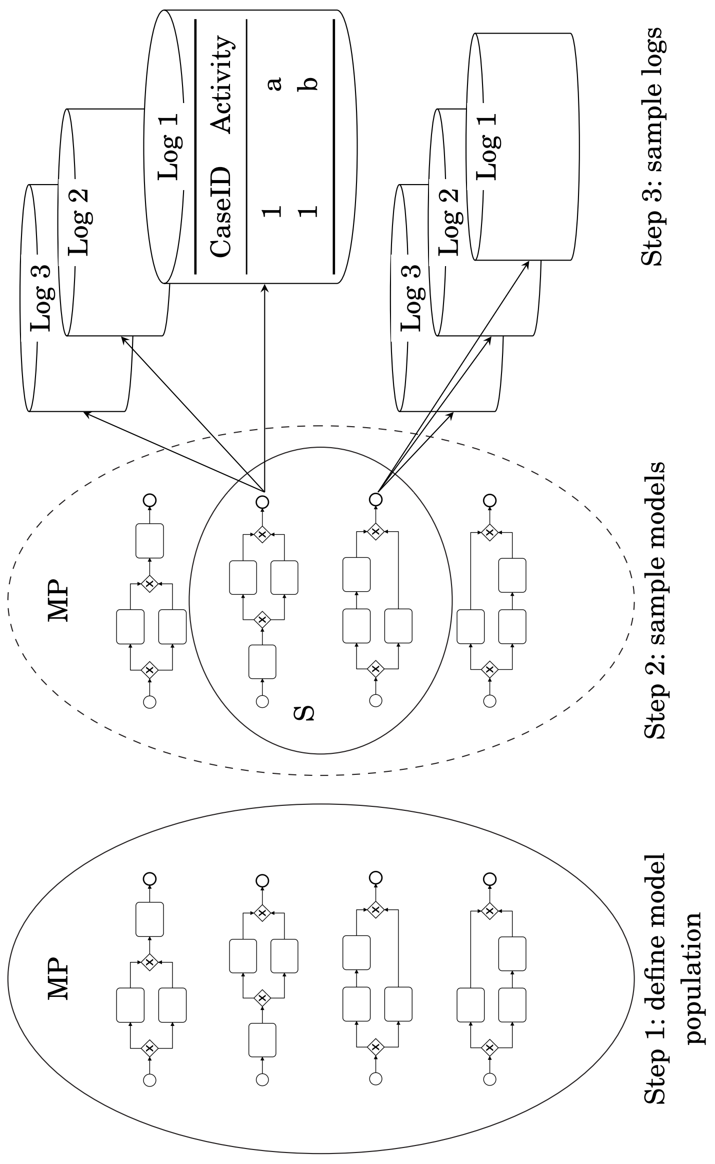


Figure 3.1: Generating artificial event data methodology.

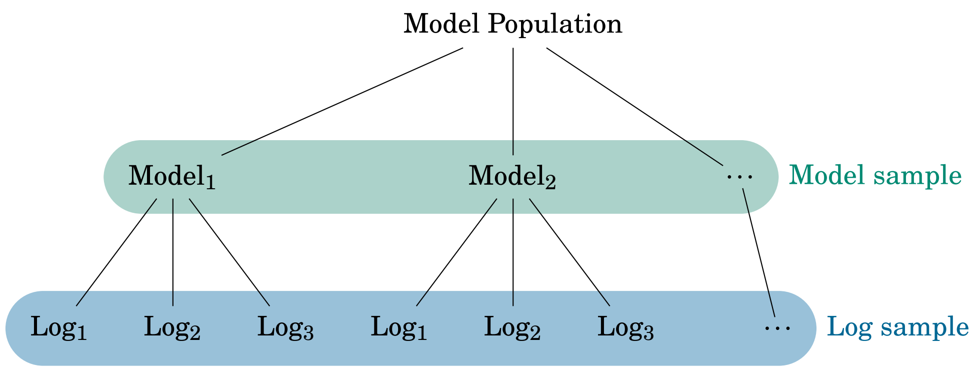


Figure 3.2: GED methodology: a hierarchical design.

event logs generated from the models in the second level. This structure gives researchers full control over the control-flow behavior in the generated event data. Additionally, it enables the researcher to generalize findings from the event logs to a known model population.

The *first step* of GED methodology comprises the definition of the model population. A model population specifies the control-flow patterns and their probabilities. Examples of the control-flow patterns are the workflow control-flow patterns (WCP), identified by [92], which represent process behavior common to all real business processes. A probability distribution is assigned to each pattern such that the sample, drawn in the *second step*, contains random models from the population. The *third step* will simulate each model in the sample into a set of event logs while setting parameters to control the number of traces and the amount of noise. This set of logs forms a sample of all possible logs produced by the model population.

The last two steps of the GED methodology are adopted from existing process mining methodology (see [29, 115, 126, 129]). In contrast to existing approaches in which researchers typically created models by hand in an ad hoc manner, the second step of GED generates models which are random observations from a model population. This allows researchers to generalize their results to a predefined population.

The next two sections of this chapter introduce a model and log generator for the GED methodology called the “Process Tree and Log Generator” (PTandLogGenerator). The PTandLogGenerator conforms to all the GED requirements listed in Section 2.3.1.

3.2 PTandLogGenerator: random model generation

This section will describe the random model generation as part of the ‘PTandLogGenerator’. The generated models are represented in the Process Tree modelling language [16, 69, 104]. Therefore, this section firstly defines process trees, then it will describe how to characterize a model population, followed by an algorithm that draws random samples of process trees from that population, and finally, how to add random LT dependencies to a process tree.

3.2.1 Process trees

A process tree is a directed connected graph without cylces that consists of leaf nodes (activities) and operator nodes (workflow patterns) [16, 104]. There are two reasons why we choose to represent processes as process trees. Firstly, because each tree is inherently sound, i.e. it is free of anomalies such as deadlocks, dead activities and livelocks [105]. This fulfills the soundness requirement of GED (see Table 2.4). Secondly, trees can be easily built in a stepwise manner using the workflow patterns in Table 2.4. This stepwise construction (explained in Section 3.2.3) will allow us to control for the process characteristics specified in the model population.

Definition 3.1 formalizes a process tree $PT(N,r,m,c,p,b)$ used in the remainder of this chapter. It extends the definition by Buijs [16] with a parent (p) and a probability mappping function (b).

Definition 3.1 (Process Tree)**.** Let $A \subseteq \mathscr{A}$ be a finite set of activities and PT be a tree: $PT = (N,r,m,c,p,b)$, where:

- N is a non-empty set of nodes consisting of operator (N_O) and leaf nodes (N_L) such that: $N_O \cap N_L = \emptyset$
- $r \in N_O$ is the root node of the tree

- $O = \{\rightarrow, \times, \wedge, \circlearrowleft^k, \vee\}$ are the base patterns: “sequence”,“choice”,“parallel”,“loop” and “or”.
- $m : N \rightarrow A \cup O \cup \{\tau\}$ is a function mapping each node to an operator or activity, with τ representing a silent activity:

$$m(n) = \begin{cases} a \in A \cup \{\tau\}, & \text{if } n \in N_L. \\ o \in O, & \text{if } n \in N_O. \end{cases}$$

  Using the mapping function m we can denote the set of all operator nodes in a tree of a specific type as N_o where $o \in O$, e.g. the set of all choice nodes in a tree is: $N_\times = \{n \in N_O | m(n) = \times\}$.
- Let N^* be the set of all finite sequences over N then $c : N \rightarrow N^*$ is the child-relation function:
  $c(n) = \langle\rangle$ if $n \in N_L$
  $c(n) \in N^*$ if $n \in N_O$
  $c(n)_i$ denotes the child node at index i in the sequence
  such that
  - each node except the root node has exactly one parent:
    $\forall n \in N \backslash \{r\} : \exists p \in N_O : n \in c(p) \wedge \nexists q \in N_O : p \neq q \wedge n \in c(q)$;
  - the root node has no parent:
    $\nexists n \in N : r \in c(n)$;
  - each node appears only once in the list of children of its parent:
    $\forall n \in N : \forall 1 \leq i < j \leq |c(n)| : c(n)_i \neq c(n)_j$;
  - a node with a loop operator type has exactly three children such that the first child is always executed first, the second child is executed maximum $k \in \mathbb{N}$ times, each time followed by the first child, and finally the third child is executed once:
    $\forall n \in N : (m(n) = \circlearrowleft^k) \Rightarrow |c(n)| = 3$.
- $p : N \rightarrow N$ is the parent relation function:
  $p(n) = q \Leftrightarrow n \in c(q)$

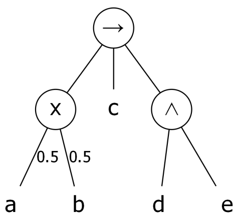


Figure 3.3: Example process tree PT_1.

- each node has a probability of being chosen:
  $b : N \to [0,1]$ is a function mapping each node n to a probability such that:

$$b(n) = \begin{cases} 1, & \text{if } p(n) \notin N_\times \\ x \in [0,1], & \text{such that } \sum\limits_{k \in c(p(n))} b(k) = 1 \text{ if } p(n) \in N_\times. \end{cases}$$

Figure 3.3 shows an example process tree PT_1 that represents a simple process that starts with a choice between activities “a” and “b”, followed by activity “c”, and then followed by activity “d” and “e” in parallel.

Definition 3.2 (Subtree)**.** Let $\sigma \cdot \sigma'$ denote the concatenation of two sequences σ and σ', then $s : N \to N^*$ is the subtree function, returning all nodes of n in a pre-order[1]:

$$s(n) = \begin{cases} m(n), & \text{if } n \in N_L. \\ m(n) \cdot \langle s(c(n)_1) \cdot \ldots \cdot s(c(n)_{|c(n)|}) \rangle, & \text{if } n \in N_O. \end{cases}$$

Applying the subtree function to the root node r_1 of process tree PT_1 produces a sequence of nodes with their labels: $s(r_1) = \to \langle \times \langle a, b \rangle, c, \wedge \langle d, e \rangle \rangle$.

The process tree operator semantics are adopted from [16]. For each operator, except for the bounded loop, there exists a trace equivalent Petri net illustrated in Table 3.1. Notice that the bounded loop is translated to an unbounded loop in the Petri net notation, but the traces in the third column are fully equivalent with the loop bounded to two iterations.

[1]Pre-order is a specific type of depth-first search as the search tree is deepened as much as possible on each child before going to the next sibling. It starts from the root node and then deepens each of its child nodes from left to right.

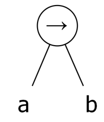
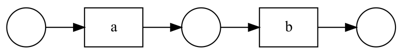
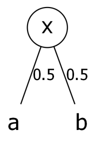
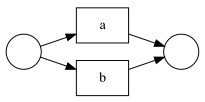
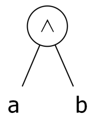
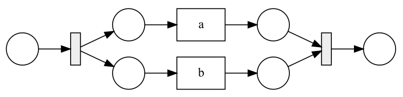
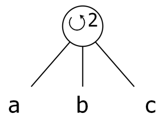
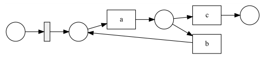
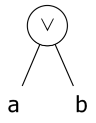
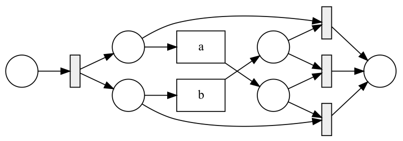
Process tree	Petri net	Trace(s)
		$\langle a,b \rangle$
		$\langle a \rangle$, $\langle b \rangle$
		$\langle a,b \rangle$, $\langle b,a \rangle$
		$\langle a,c \rangle$, $\langle a,b,a,c \rangle$, $\langle a,b,a,b,a,c \rangle$
		$\langle a \rangle$, $\langle b \rangle$, $\langle a,b \rangle$, $\langle b,a \rangle$

Table 3.1: Illustration of process tree translation to Petri net with possible traces.

3.2.2 Define a model population

The first step of the GED methodology is the definition of a model population. In this step the user defines the building blocks, i.e. control-flow patterns, of which the models in the population consist. The full control requirement category in Table 2.4 listed an extensive set of control-flow patterns a researcher wants to control during discovery algorithm evaluation (listed in the first column of Table 3.2).

In the 'PTandLogGenerator', the model population consists of process trees. Each process tree consists of operator and leaf nodes (see Definition 3.1). An operator node represents one of the basic workflow control-flow patterns: "sequence" ($\rightarrow$), "choice" ($\times$), "parallel" ($\wedge$), "loop" ($\circlearrowleft^k$) and "or" ($\vee$). A leaf node represents an activity which can be a visible activity ($a \in A$) or a silent activity (τ).

Our method requires users to specify six probability distributions to define a tree population. First, the user assigns a triangular distribution for the number of visible activities to control the size of the trees. A triangular distribution is characterized by a lower limit, a mode and an upper limit: y $\sim$ triangular(minimum,mode,maximum). As a result, all trees in the population will have a number of visible activities $y = |\{n \in N_L | m(n) \in A\}|$ between the lower and upper limit with the mode as most likely value.

Secondly, the frequency of the operator types "sequence" ($\rightarrow$), "choice" ($\times$), "parallel" ($\wedge$), "loop" ($\circlearrowleft^k$) and "or" ($\vee$) in a tree is defined by a categorical distribution. As a result, each of these operator types has a fixed probability: $\Pi^{\rightarrow}, \Pi^{\wedge}, \Pi^{\times}, \Pi^{\vee}, \Pi^{\circlearrowleft}$. Together the probabilities of these basic patterns should always sum to one.

Finally, each of the more complex patterns are assigned to a binomial distribution. The binomial distribution is used as each of those patterns is added in a series of "yes/no" questions. The number of silent activities depends on the probability Π^{τ} to add a silent activity to a "choice" or "loop" node. The number of reoccurring activities is determined by the probability to duplicate a visible activity Π^{Re}. The number of LT dependencies is subject to the likelihood Π^{Lt} of inserting a dependency between activities in "choice" nodes. Finally, the number of "choices" with infrequent outgoing path(s) depends on the probability Π^{In}.

Definition 3.3 formalizes a model population MP used in the remainder of this thesis.

Definition 3.3 (Model Population)**.** A model population is defined as $MP =$ (minimumVisibleAct, modeVisibleAct, maxVisibleAct, Π^{Base},Π^{τ},Π^{Re},Π^{Lt},Π^{In}) such that:

- The minimumVisibleAct, modeVisibleAct, maxVisibleAct parameters specify the size of the process trees in population in terms of the number of visible activities y that follows a triangular distribution between minimumVisibleAct (lower limit) and maxVisibleAct (the upper limit) with modeVisibleAct the most likely value: $y \sim$ triangular(minimumVisibleAct, modeVisibleAct, maxVisibleAct).

- Π^{Base} denotes the set of fixed probabilities of the basic tree operator types, i.e.
$$\Pi^{Base} = \{\Pi^{\rightarrow}, \Pi^{\wedge}, \Pi^{\times}, \Pi^{\vee}, \Pi^{\circlearrowleft}\}$$
. The type of an operator node $n \in N_O \sim$ Categorical(Π^{Base}).

- Π^{τ} specifies the probability of silent activity in a "choice" and "loop" node of a process tree. The number of silent activities in a process tree follows a Binomial distribution, i.e. the distribution of the number of times "yes" is answered in a sequence of independent questions "add silent activity?" for each "choice" and "loop" node of a tree: number of silent activities $\sim$ Binomial($|\{N_{\times} \cup N_{\circlearrowleft}\}|, \Pi^{\tau}$). Notice that the number of silent activities depends on $\Pi^{\times}$ and $\Pi^{\circlearrowleft}$.

- Π^{Re} specifies the probability of duplicating a visible activity in a process tree. The number of duplicated visible activities in a process tree follows a Binomial distribution, i.e. the distribution of the number of times "yes" is answered in a sequence of independent questions "change visible activitiy to invisible?" for each visible activity in a tree: number of duplicated visible activities $\sim$ Binomial(y, Π^{Re}) with y the number of visible activities.

- Π^{Lt} specifies the probability of inserting a long-term dependency in a process tree. The number of LT dependencies follows a Binomial distribution, i.e. the distribution of the number of times "yes" is answered in a sequence of independent questions "add LT dependency?" for each possible point for LT dependency insertion (see Section 3.2.4): number of LT dependencies $\sim$ Binomial(Δ, Π^{Lt}) with Δ the total number of possible dependencies which depends on $\Pi^{\times}$ as explained in Section 3.2.4.

- Π^{In} specifies the probability of a "choice" node in a tree having infrequent outgoing paths. The number of choice nodes with infrequent paths follows a Binomial distribution, i.e. the distribution of the number of times "yes" is answered in a sequence of independent questions "add infrequent paths?" for each choice node in a process tree: number of choice nodes with infrequent paths $\sim$ Binomial($|N_{\times}|, \Pi^{In}$). Notice that the number of choice nodes with infrequent paths depends on $\Pi^{\times}$.

Parameter	**Setting**
Number of Visible Activities	(min,mode,max)
Sequence ($\Pi^{\rightarrow}$) (WCP-1)	$\in [0,1]$
Parallel ($\Pi^{\wedge}$) (WCP-2/3)	$\in [0,1]$
Choice ($\Pi^{\times}$) (WCP-4/5)	$\in [0,1]$
Loop ($\Pi^{\circlearrowleft}$) (WCP-21)	$\in [0,1]$
Or ($\Pi^{\vee}$) (WCP-6/7)	$\in [0,1]$
Silent activities (Π^{τ})	$\in [0,1]$
Reoccurring activities (Π^{Re})	$\in [0,1]$
Long-term dependencies (Π^{Lt})	$\in [0,1]$
Infrequent paths (Π^{In})	$\in [0,1]$

Table 3.2: Probability settings of control-flow patterns

A model population actually represents an infinite set of models with characteristics defined by the probabilities in its definition MP. From an even higher level of abstraction, the model population is a sample of all possible model populations, i.e. a set of all possible distributions over process characteristics.

Table 3.2 provides an overview of the probabilities and the valid setting that a user needs to determine for each element in the model population MP.

3.2.3 Sample models

The definition of the model population enables the second step of the GED methodology: draw a random sample of models from the population. The implementation of this step uses a process tree generating algorithm illustrated in Figure 3.4.

The algorithm is a top-down approach based on the random node addition method for the mutation of process trees presented by [16]. Unlike that method, which generates completely random trees, the presented algorithm generates process trees that are random observations of a predefined model population. This allows researchers to control the behavior in the models and generalize their results to a known model population.

The tree building algorithm in Figure 3.4 builds a random process tree PT given a model population MP. It starts by drawing a random value y from the distribution of activities to decide how large the tree will grow in terms of visible activities. After that, the algorithm adds nodes to the tree for as long as there are activities left to incorporate ($\#act < y$). In each iteration the algorithm selects a random visible leaf node (or the root node in case the tree has no nodes yet) and replaces this node with an operator node based on the probabilities in Π^{Base}.[2] Then, the algorithm adds leaf nodes to the assigned operator: a loop node always has three leaf nodes, all the other operators get two leaf nodes. If the operator is of type choice or loop, one of the added leaf nodes can be an invisible activity based on the probability Π^{τ}. The next step updates the number of visible activities $\#act$ in the tree. After all activities are added ($\#act = y$), the tree is reduced. This step merges parent and child nodes if they have the same operator type except for loops.[3] As a result, the reduced tree is not limited to operators with only two children. The next step of the algorithm duplicates the labels of leaf nodes based on the probability Π^{Re}. Finally, the algorithm assigns

[2]Invisible activities are endpoints in the tree and hence are never selected to be replaced.

[3]Reducing parent and child loop nodes could cause the parent loop node to have more than three children.

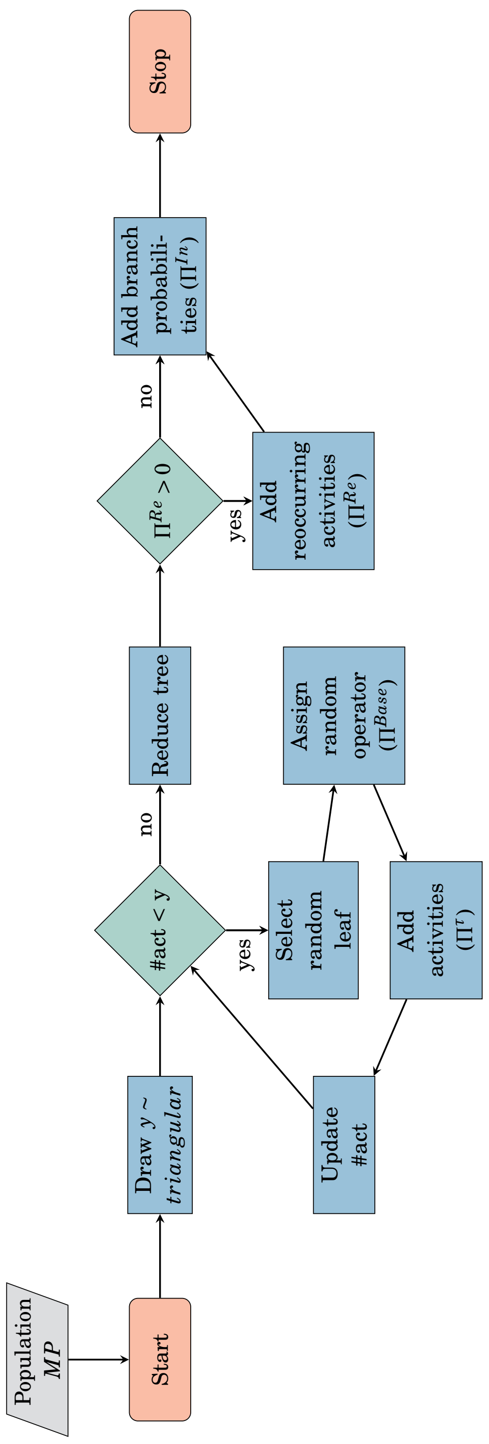

Population MP
Start
Draw $y \sim triangular$
$\#act < y$
yes
no
Select random leaf
Assign random operator (Π^{Base})
Add activities (Π^{ι})
Update #act
Reduce tree
$\Pi^{Re} > 0$
yes
no
Add reoccurring activities (Π^{Re})
Add branch probabilities (Π^{In})
Stop

Figure 3.4: Flowchart of tree building algorithm

either equal or unequal branch probabilities to each choice node based on the probability Π^{In}.

Figure 3.5 illustrates the tree building algorithm. Suppose that the model population is as follows: $MP = (min = 5, mode = 7, max = 10, \Pi^{Base} = \{0.4, 0.2, 0.2, 0, 0.2\}, 0.3, 0.1, 0.0, 0.5)$. The algorithm starts by drawing $y = 7$ as the number of visible activities in the final tree. Then, as the tree has no nodes yet, i.e. $\#act = 0$, the algorithm starts adding nodes to the tree. It starts with assigning a random operator to the root node. Suppose this is a sequence operator. Then, two activities are added as leaf nodes "a" and "b" to the root node (see Figure 3.5(a)). The number of activities is updated to $\#act = 2$. As $\#act < 7$, the algorithm selects a random leaf node, suppose leaf node "b". Next, it replaces "b" by a randomly selected operator, e.g. a choice operator. This choice operator gets two activities "b" and "c" as leaf nodes (see Figure 3.5(b)) which updates the number of activities to $\#act = 3$. Similarly, leaf node "a" is replaced by a sequence operator with "a" and "d" as children (see Figure 3.5(c)). Next, the algorithm replaces leaf node "d" with a loop operator. For a loop operator, the middle child can contain an invisible activity depending on the user specified probability Π^{τ}. Suppose, that a randomly drawn number is smaller than Π^{τ} such that the algorithm adds an invisible activity as middle child (see Figure 3.5(d)). The next two node additions extend the tree with a parallel and a sequence operator (see Figure 3.5(e) and 3.5(f)) to reach the target number of activities, i.e. $\#act = 7$. Then, the algorithm reduces the tree by merging the two sequence operators (see Figure 3.5(g)). Next, the algorithm iterates over all the visible activities in the tree and randomly changes the label of a visible activity to the label of a randomly chosen other visible activity based on the probability $\Pi^{Re} = 0.3$, in this example the label of leaf node "c" is changed to "a" in Figure 3.5(h). Finally, the algorithm iterates over all choices in the tree and randomly assigns unequal branching probabilities to the choice based on the probability $\Pi^{In} = 0.5$. In this example the only choice operator in the tree gets unequal branching probabilities: 0.1 and 0.9 (see Figure 3.5(h)).

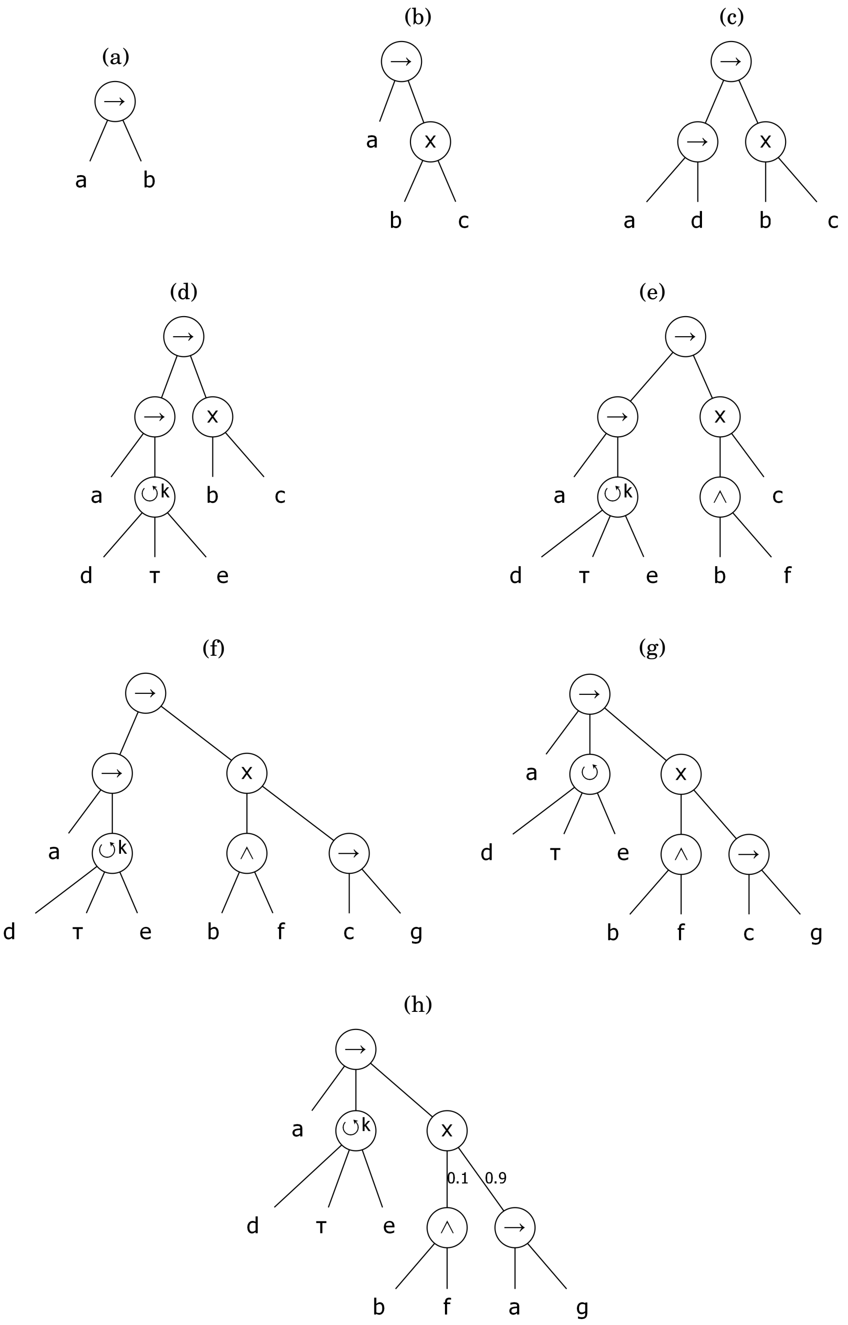


Figure 3.5: Illustration of the tree building algorithm

A process tree generated by the above algorithm is free-choice and therefore does not contain LT dependencies. However, the requirements of GED in Table 2.4 specified that the random model generator should also enable LT dependencies. Therefore, the next section presents a method to add random LT dependencies to a given process tree.

3.2.4 Adding long-term dependencies

Process trees, as generated in the previous step, are block-structured models. As a result, all dependencies in a tree are local, i.e. there are no LT dependencies. Previous approaches that generate models with LT dependencies do not guarantee soundness, another requirement of the GED methodology. Therefore, we propose an approach to incorporate random LT dependencies in a given tree resulting in a so called "unfolded choice tree" which is always sound (see Algorithm 1). We adopt the definition of LT dependencies of [16]: "choices that depend on decisions made earlier in the process". It focuses on decisions represented as exclusive choices (WCP-4 and 5), as such the considered LT dependencies correspond to the non-free-choice constructs in cases a,e,f and g of Figure 5 in [129] and shown here in Figure 3.6 for ease of reference. To our knowledge, this approach is the first to extend process trees with LT dependencies.

As an example, consider the process tree PT_2 illustrated in Figure 3.7. Tree PT_2 has a "sequence" operator as root node with several "choice" nodes (choices) as descendants.[4] PT_2 contains no LT dependencies, e.g. if activity "a" was chosen in $\times\langle a,b\rangle$, then this decision would not affect the choice between "f" and "g" in $\times\langle f,g\rangle$ later in the process. The proposed approach allows to incorporate LT dependencies between choices. Consider for example a dependency between activities in choices $\times\langle a,b\rangle$ and $\times\langle f,g\rangle$: if "a" is chosen, then "f" cannot be chosen later on.

The following paragraphs will describe the two steps of the proposed approach to insert LT dependencies: a tree preparation step followed by an insertion step.

[4] A descendant is a node reachable by repeatedly going from parent to child.

Algorithm 1 :Insert random long-term dependencies

```
Input:
    $PT$: process tree
    $\Pi^{Lt}$: probability of inserting a LT dependency
    $UnfoldLoops$: whether or not to unfold loops
    $k$: maximum repetitions of loops
Output:
    $PT^{\times}$:unfolded choice tree with dependencies
Start InsertRandomLTDependencies($PT, \Pi^{Lt}$,
        $\boldsymbol{UnfoldLoops}, k$)
$PT^{\times} \leftarrow PT$
while $\exists n' \in s(r') | n' \neq r'$ $and$ $n' \in N'_{\times}$ do
    if $UnfoldLoops$ = True then
        apply transformation rule to $n'$ or unfold loop with maximum $k$ repetitions
    else
        apply transformation rule to $n'$
    end if
    move the branching probabilities of $n'$
end while
for $n' \in c(r')$ do
    $x \leftarrow random$, $i \leftarrow$ index of $n'$ in $c(r')$
    if $x < \Pi^{Lt}$ and $\phi(PT^{\times}, i)$ = true then
        remove entire branch $s(n')$ from $PT^{\times}$
    end if
end for
$z \leftarrow \sum_{n \in c(r')} b(n)$
for $n \in c(r')$ do
    $b(n') \leftarrow b(n')/z$
end for
return $PT^{\times}$
```

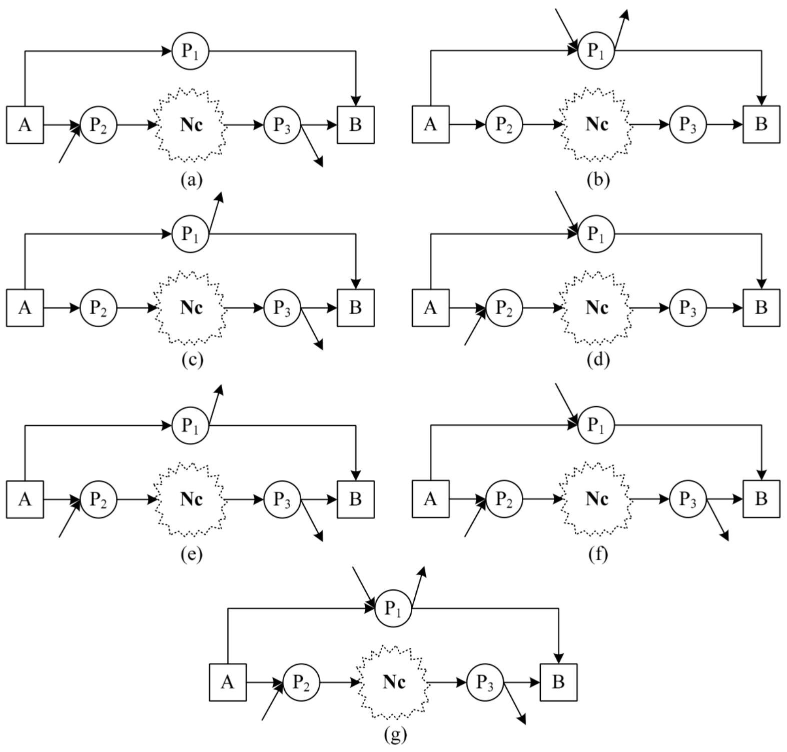


Figure 3.6: Figure 5 in [129] illustrating possible non-free-choice constructs in Petri nets.

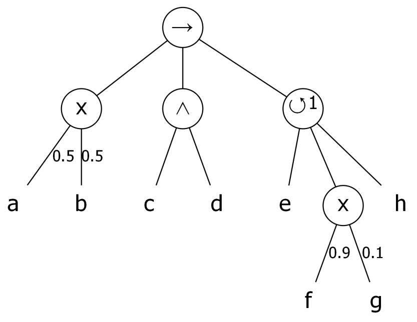


Figure 3.7: Example process tree PT_2.

3.2.4.1 Preparing the Tree for Long-term Dependencies

The first step of the approach to insert LT dependencies is a preparation step. A LT dependency limits the choice behavior of one choice based on what happened in (an)other choice(s). For example, a dependency between activity “a” and “f” in Figure 3.7 limits the behavior in choice $\times\langle f,g\rangle$, i.e. if “a” happens, then “f” cannot be chosen in $\times\langle f,g\rangle$. As such, a LT dependency forbids behavior in a combination of choices of a tree.

To insert LT dependencies in a process tree, one needs combinations of choice behavior. These combinations arise if a process tree has multiple choice operators and a combination represents one outgoing branch from every choice operator in the tree. For example, process tree PT_2 in Figure 3.7 has the following combinations of choice behavior: “a” and “f”, “a” and “g”, “b” and “f”, and “b” and “g”. However, a process tree in its normal form does not display such combinations. Therefore, the proposed approach first transforms the given tree $PT(N,r,m,c,p,b)$ into a trace equivalent tree called the unfolded choice tree using duplication of activity labels. The transformed tree, denoted as $PT^{\times}(N',r',m,c,p,b)$, contains only one choice which is the root node r'. Each branch (subtree) under the root r' contains a combination of choice behavior in the original tree. As such, the choice at the root r' represents all choices in the original tree. At the same time, $PT^{\times}$ is still block-structured and thus sound (see proof of Theorem 3.1).

Lines 10-18 of Algorithm 1 describe how to unfold the original tree PT into $PT^{\times}$ in a recursive way using the transformation rules in Definition 3.4. Each time, take the deepest choice in the tree and apply a transformation rule in Definition 3.4 to move it closer to the root node. Notice that there is no transformation rule for a loop node with a choice as the first or second child. Directly unfolding a loop with a choice in the first or second child of a loop would make $PT^{\times}$ not trace equivalent to PT.[5] Therefore, the user can decide if such choices in loops are unfolded. Not unfolding these first and second child choices will exclude them from the generated LT dependencies. If a user chooses to

[5]Tree $s(PT_1) = \circlearrowleft^k \langle \times\langle a,b\rangle,c,d\rangle$ is not trace equivalent to tree $s(PT_2) = \times\langle \circlearrowleft^k \langle a,c,d\rangle, \circlearrowleft^k \langle b,c,d\rangle\rangle$.

unfold the choices in the first or second child of the loop, then this requires a special unfolding step for that particular loop.

Definition 3.1 specifies that a loop node has exactly three children such that the first child node is always executed first, the second node is executed maximum k times, each time followed by the first child node, and finally the third child node is executed to conclude. Because k is a finite number, one can unfold the bounded loop into a trace equivalent structure of $\rightarrow$ and $\times$ nodes as illustrated in Figure 3.8 with $k = 2$. The bounded loop can be justified by accepting a so-called fairness assumption by [98]: "soundness and strong fairness means that each process instance will eventually terminate correctly". The user can specify the number k, i.e. the maximum times a loop can repeat. After the loop unfolding, the resulting choices can be unfolded again with the rules in Definition 3.4.

When applying the transformation rules in Definition 3.4 the branching probabilities of the children of the original choice move to the children of the new unfolded choice. The loop unfolding results in a choice node with as children the number of loop repetitions. The probability of these repetitions is defined using a categorical distribution: $\Pi^{Repetitions} = \{\Pi^0, \Pi^1, \ldots, \Pi^{k-1}, \Pi^k\} : \Pi^i = 0.5^{i+1}\ \forall i \in [0, k-2]$ and $\Pi^i = 0.5^k\ \forall i \in [k-1, k]$, where Π^i is the probability of i repetitions and k the maximum number of repetitions. As such this distribution is equivalent to the behavior of a bounded loop with a probability of 50% to do a loop iteration and a probability of 50% to exit the loop.

Definition 3.4 (Unfolded Choice Tree)**.** A given tree $PT = (N, r, m, c, p, b)$ with at least one choice block, i.e. $|\{\times_i | \times_i \in N_O\}| \geq 1$, can be transformed to the unfolded choice tree form $PT^{\times} = (N', r', m, c, p, b)$ using the following rules:

1. $\rightarrow(\times(\ldots_1, \ldots_2), \ldots_3) = \times(\rightarrow(\ldots_1, \ldots_3), \rightarrow(\ldots_2, \ldots_3))$

2. $\times(\times(\ldots_1, \ldots_2), \ldots_3) = \times(\ldots_1, \ldots_2, \ldots_3)$

3. $\wedge(\times(\ldots_1, \ldots_2), \ldots_3) = \times(\wedge(\ldots_1, \ldots_3), \wedge(\ldots_2, \ldots_3))$

4. $\circlearrowleft^k(\ldots_1, \ldots_2, \times(\ldots_3, \ldots_4)) = \times(\circlearrowleft^k(\ldots_1, \ldots_2, \ldots_3), \circlearrowleft^k(\ldots_1, \ldots_2, \ldots_4))$

5. $\vee(\times(\ldots_1, \ldots_2), \ldots_3) = \times(\vee(\ldots_1, \ldots_3), \vee(\ldots_2, \ldots_3))$

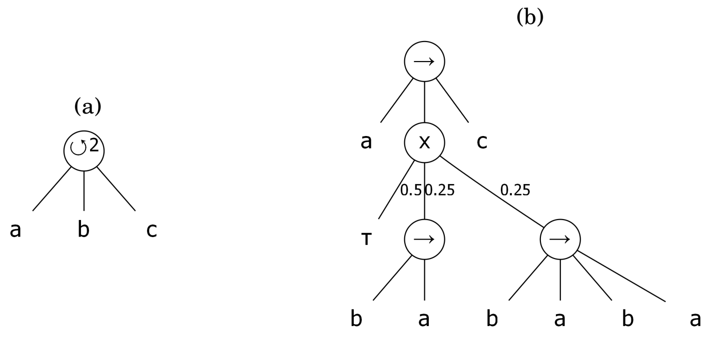


Figure 3.8: Illustration of the loop unfolding step. Left: the original bounded loop. Right: the unfolded loop with maximum 2 iterations.

The branching probabilities assigned to each of the children of a choice node $\times_i$ by the mapping function $b(c(\times_i)_j)$ in PT are transferred to the new choice $\times_i'$ in $PT^\times$ each time a rule is applied:

- if $p(\times_i) \in \{N_\rightarrow \cup N_\wedge \cup N_\vee \cup N_\circlearrowleft\}$, then the probabilities move up with the $\times_i$ operator: $b(c(\times_i')_j) = b(c(\times_i)_j)$
- if $p(\times_i) \in N_\times$, then the branching probabilities of both choice nodes are multiplied when merging:
  $\times_p(\times_i(\ldots_1,\ldots_2),\ldots_3) = \times_{pi}'(\ldots_1,\ldots_2,\ldots_3)$, then the probabilities of $\times_{pi}'$ are:
  - $b(c(\times_{pi}')_1) = b(c(\times_p)_1) \cdot b(c(\times_i)_1)$
  - $b(c(\times_{pi}')_2) = b(c(\times_p)_1) \cdot b(c(\times_i)_2)$
  - $b(c(\times_{pi}')_3) = b(c(\times_p)_3)$

To illustrate the tree transformation, consider the tree PT_2 in Figure 3.9(a). First, select the deepest choice node that is not the first or second child of a loop node, i.e. $\times\langle a,b\rangle$. Then apply transformation rule 1 of Definition 3.4 to obtain the tree in Figure 3.9(b). This tree contains two branches that are equal, except for the leaf nodes “a” and “b”. The probabilities of the original children of the choice, i.e. “a” and “b”, move up together with the choice operator: the probability to execute the left (right) branch of the top choice node is 0.5, which is the probability to execute “a” (“b”) of the original choice $\times\langle a,b\rangle$ before applying the transformation rule.

The two remaining choices under the root are both the second child of a loop node. To include these choices in LT dependencies, a loop unfolding step is needed. Consider for example an unfolding with a maximum of 1 repetition[6], then the trace equivalent unfolded tree is shown in Figure 3.9(c).

Due to the unfolding of the loops, new choice nodes appear under the root node. Therefore, similarly to the first step, one can again apply transformation rule 1 to the choice $\times\langle f,g\rangle$ (in the left and right branch) to obtain the tree in Figure 3.9(d). Notice that the probabilities to execute branches “f” (0.9) and “g” (0.1) move to the left and right branches of the transformed choice. In the next step, apply transformation rule 2 to merge parent with child choices. This results in the tree in Figure 3.9(e). The probabilities of parent and child branches are multiplied when merging the choices: the probability of τ remains at 0.5, but the probability of the right parent branch is multiplied with the probabilities of the two child branches, i.e. $0.5 \cdot 0.9 = 0.45$ is the probability of the merged branch $\rightarrow\langle f,e\rangle$ and $0.5 \cdot 0.1 = 0.05$ is the probability of the other merged branch $\rightarrow\langle g,e\rangle$.

The unfolding of all choices continues until the root node of the tree is the only choice node in the tree (e.g. in Figure 3.10). If the user opts to not include the choices in the first or second child of a loop node in the dependencies, the unfolding stops when the root node is a choice and all other choices are either a first or second child under a loop node (e.g. in Figure 3.9(b)).

[6]Notice that activity “e” can be repeated once.

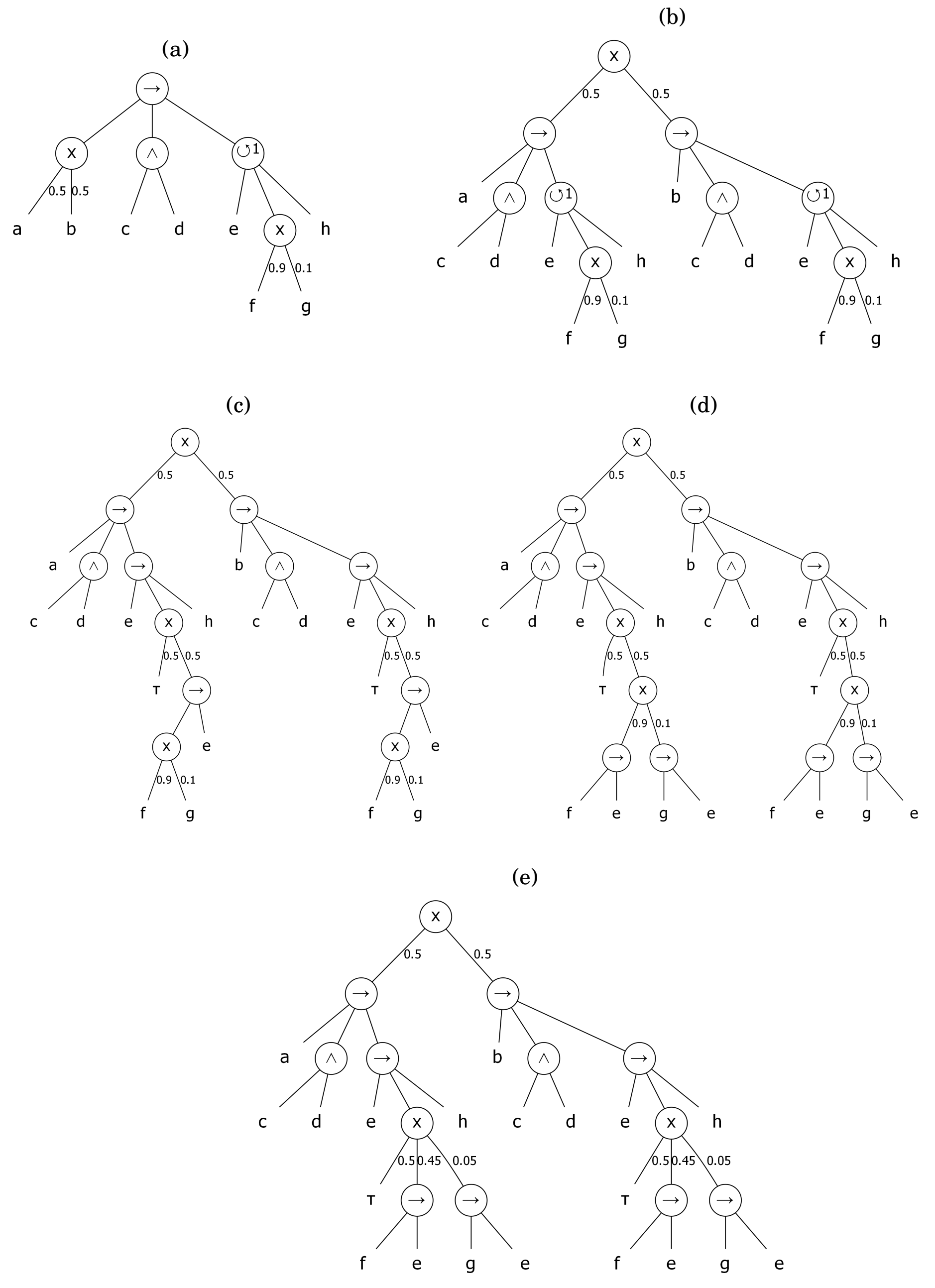


Figure 3.9: Unfolding of Tree PT_2: (a) original tree PT_2, (b) unfolded choice rule 1, (c) unfolded loops, (d) unfolded choice rule 1, (e) merged choices.

3.2.4.2 Inserting Random Dependencies

After the preparation step, the approach inserts random LT dependencies into the unfolded choice tree (see lines 19-24 of Algorithm 1). LT dependencies are created on a trace level while ensuring soundness. Each branch under the root node of the unfolded choice tree $PT^{\times}$ represents one combination of choice behavior in the original tree PT, i.e. a set of traces in the resulting event log. Removing a branch from the tree $PT^{\times}$ forbids this combination and thus inserts a LT dependency. To ensure random LT dependencies, the removal of a branch depends on the probability to insert LT dependencies Π^{Lt}.

To guarantee soundness, one could not simply remove any set of combinations of choice behavior, because some combinations together restrict too much choice behavior and thus result in dead activities. A dead activity occurs if an activity in the original tree PT does not occur in the tree $PT^{\times}$. The goal of the approach is to insert LT dependencies by limiting the choice behavior in a tree while preventing unsound behavior such as dead activities.

The pruning mechanism in Definition 3.5 prevents dead activities by checking if removing a branch from the root causes a dead activity in tree $PT^{\times}$. First, the mechanism retrieves all activities in the branch A_i. Then, it retrieves all activities in the other branches: A_o. If the activities in the selected branch are not contained in the set of activities of the other branches, i.e. $A_i \nsubseteq A_o$, then the selected branch cannot be removed.

Definition 3.5 (Pruning Mechanism)**.** The pruning mechanism is a function $\phi : PT^{\times} \times \mathbb{N} \rightarrow [\text{true}, \text{false}]$ that given the unfolded choice tree $PT^{\times}$ and the index i of a branch of the root returns "false" if a dead activity occurs when eliminating the branch $c(r')_i$ in $PT^{\times}$:

$$A_i = \{m(n') \in N'_L | n' \in s(c(r')_i)\}$$
$$A_o = \{m(n') \in N'_L | n' \notin s(c(r')_i)\}$$

$$\phi(PT^{\times}, i) = \begin{cases} \text{true}, & \text{if } A_i \subseteq A_o. \\ \text{false}, & \text{if } A_i \nsubseteq A_o. \end{cases}$$

The insertion of LT dependencies is illustrated in Figures 3.10, 3.11, 3.12, 3.13. It starts from the unfolded choice tree $PT_2^{\times}$ (see Figure 3.10) obtained from unfolding PT_2 in Figure 3.9. Then, Algorithm 1 visits each of the branches under the root choice node. Based on the probability to insert LT dependencies, $\Pi^{Lt} = 0.5$, the first, third and last branch are randomly selected as candidates for removal. The first and third branch can be removed as illustrated in Figure 3.11 and Figure 3.12 respectively. However, the pruning mechanism prevents removing the last branch as this would make "g" a dead activity.

Finally, after removing the branches in $PT^{\times}$, the sum of the branching probabilities of the remaining children of the root does not equal to one: i.e. $\sum_{n \in c(r')} b(n) \neq 1$. Therefore, the branching probabilities of each of these child nodes are normalized (see lines 25-28 of Algorithm 1): for each node $n_i \in c(r')$ do $b(n) = b(n_i) / \sum_{n \in c(r')} b(n)$. In the example the branching probabilities of the tree in Fig. 3.12 are normalized as shown in Fig. 3.13. This results in the final unfolded choice tree with LT dependencies which is sound:

Theorem 3.1. *Algorithm 1 generates unfolded choice trees with long-term dependencies that are sound.*

Proof. The proposed algorithm generates LT dependencies on a trace level. It first transforms the original tree in a trace equivalent unfolded choice tree to control that the long-term depency insertion does not alter other process behavior. The algorithm removes a set of branches from the unfolded choice tree to exclude some combinations of choice behavior in the tree. In this way choices are no longer free, but depend on other choices made earlier in the process, which conforms to the definition of *LT dependencies* (see introduction of Section 3.2.4).

The unfolded choice tree $PT^{\times}$ is created by applying the five transformation rules in Definition 3.4 and the loop unfolding step.

The transformation rules use the operators $O = \{\rightarrow, \times, \wedge, \circlearrowleft^k, \vee\}$ and add duplicate activity labels, i.e. $m(n_1') = m(n_2') | n_1' \neq n_2'$ and $n_1', n_2' \in N_L'$, to unfold choices. Each of those transformation rules preserves the language of the original process tree. The language of a process tree is defined as the set of all completed

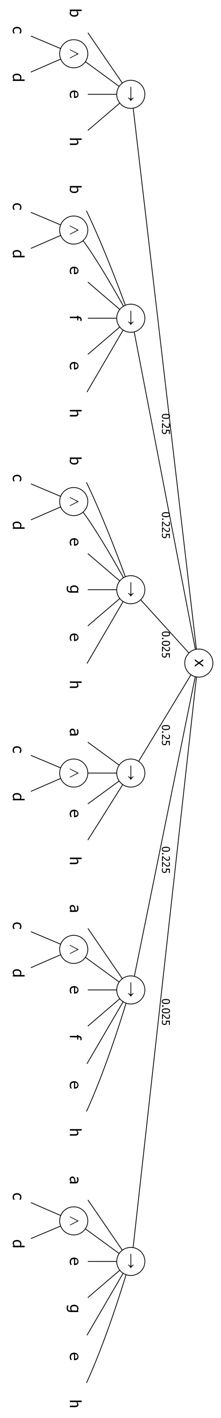


Figure 3.10: Unfolded choice tree $PT_2^{\times}$

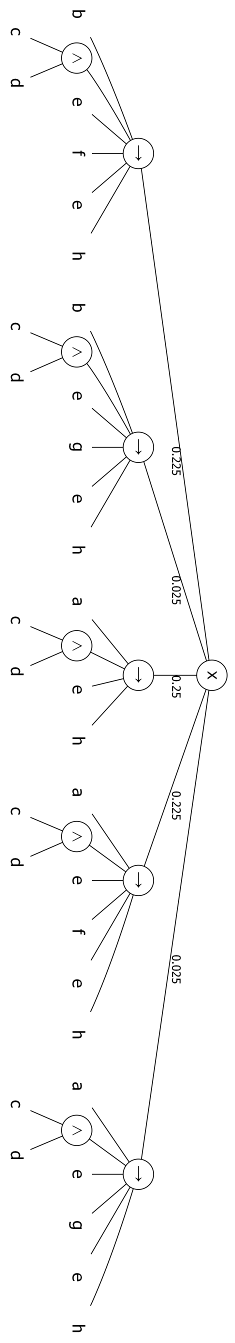


Figure 3.11: Unfolded choice tree $PT_2^{\times}$ after removal of the first branch

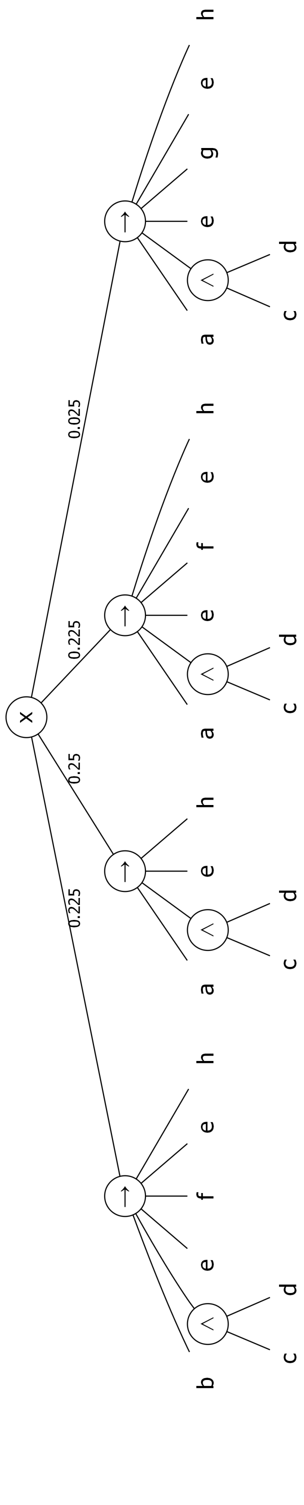


Figure 3.12: Unfolded choice tree $PT_2^\times$ after removal of the first and third branch

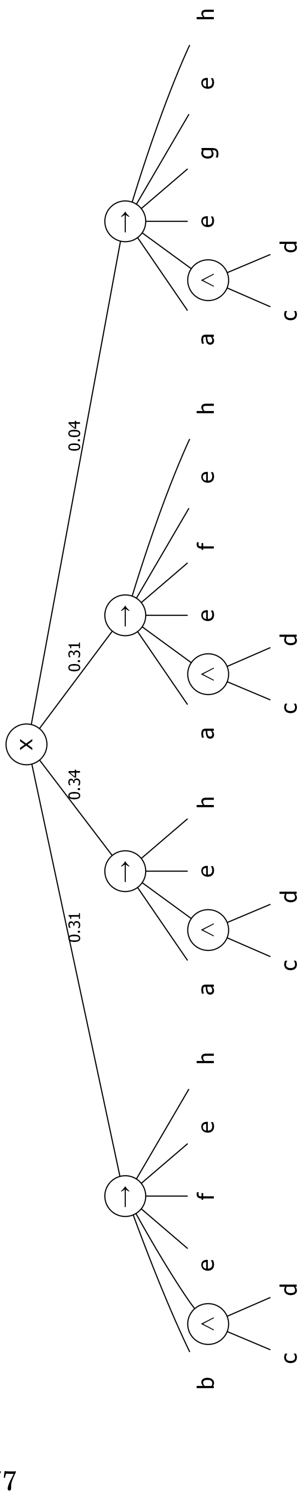


Figure 3.13: Normalized branching probabilities

traces that the tree can produce. Table 3.1 illustrates the language of each process tree operator. The formal language of each operator is formulated in Defition C.2 in Appendix C. If a transformed tree has the same language as the original tree, then this means that it ensures trace equivalent behavior. To illustrate this, consider a process tree PT_3: $s(r_3) = \rightarrow \langle \times \langle a, b\rangle, c\rangle$. After applying transformation rule 1 of Definition 3.4 to PT_3 one gets $PT_3^\times$: $s(r_3^\times) = \times\langle \rightarrow \langle a, c\rangle, \rightarrow \langle b, c\rangle\rangle$. It is easy to see that the language of PT_3 and $PT_3^\times$ is the same: $\mathscr{L}(PT_3) = \mathscr{L}(PT_3^\times) = \{\langle a, c\rangle, \langle b, c\rangle\}$.

The loop unfolding step replaces a loop operator ($\circlearrowleft^k$) by a combination of sequence and choice operators ($\rightarrow$, $\times$) plus a silent activity (τ) and duplicate activity labels. The unfolded loop and the original bounded loop have the same language which ensures trace equivalent behavior. As an example, consider PT_4: $s(r_4) = \circlearrowleft^2 \langle a, b, c\rangle$ and the tree after unfolding the loop $PT_4^\times$: $s(r_4^\times) = \rightarrow \langle a, \times\langle \tau, \rightarrow \langle b, a\rangle, \rightarrow \langle b, a, b, a\rangle\rangle, c\rangle$. It is easy to see that the language of PT_4 and $PT_4^\times$ is the same: $\mathscr{L}(PT_4) = \mathscr{L}(PT_4^\times) = \{\langle a, c\rangle, \langle a, b, a, c\rangle, \langle a, b, a, b, a, c\rangle\}$.

As such the transformation rules plus the loop unfolding results in an unfolded choice tree that conforms with the Definition 3.1 of a block-structured process tree which is inherently block-structured and thus *sound*.

The removal of branches (subtrees) of the unfolded choice tree $PT^\times$ can never introduce deadlocks as each branch under the root node in $PT^\times$ is sound and independent from other branches, i.e. they are mutually exclusive, and removing one branch does not affect the remaining branches. However, the removal can produce dead activities by eliminating all branches in which a certain activity occurs. The pruning mechanism in Definition 3.5 prevents dead activities by ensuring that each activity occurs in at least one branch of the final tree. The absence of dead activities together with the block-structuredness of $PT^\times$ guarantees *soundness*. ■

The following section will discuss how the generated trees can be simulated into event logs.

3.3 PTandLogGenerator: random event log generation

This section focuses on the last step of the GED methodology: how to generate a sample of event logs from a sample of trees as generated in Section 3.2.

3.3.1 Setting Log Characteristics

The hierarchical design of the GED methodology (see Fig. 3.2) shows that one process tree represents a population of event logs. The population can be further refined using log characteristics. Here we use two characteristics imposed by the full-control requirement in Table 2.4: the number of traces and the amount of noise. Similar to the model population, the user needs to specify each of these log characteristics: a fixed number for the number of traces and a distribution of noisy traces as described below.

Definition 2.3 formalized a simple trace as a sequence of activities and a simple event log as a multiset of traces. The size of the log $|L|$ is equal to the number of traces t. It expresses how many times the simulator will run from start to end through the process tree, logging each of these runs as a separate trace σ_j.

This thesis adopts the definition of noise by Günther [41]: “noise is incorrect behavior in the log that can be caused either by the logging mechanism or the constitution of the event data". The following types of noise behavior are adopted from [41]: missing head, missing body (episode), missing tail, order perturbation and the introduction of additional activities. Often during evaluation, these noise types are introduced in a log as they typically pose problems to discovery algorithms as they can lead to, respectively: erroneous start activities, incorrect activity skips, erroneous end activities, incorrect parallellism, and erroneous loops on an activity. Assume a trace $\sigma_j = \langle a_1, \ldots, a_{n-1}, a_n \rangle$. The missing head, body and tail types, remove subsequences of a trace σ_j. The head of a trace contains activities a_i with $i \in [1, n/3]$, the body consists of activities a_i with $i \in [(n/3)+1, 2n/3]$ and the tail contains activities a_i with $i \in [(2n/3)+1, n]$. The order perturbation type interchanges two random activities. The additional activities type introduces a random activity from the available alphabet in the trace.

The amount of noisy traces t^* in a log is specified using a binomial distribution: $t^* \sim \text{Binomial}(|L|, \Pi^{Noise})$. Π^{Noise} expresses the probability to select a trace for noise insertion. A noisy trace contains a random type of noise behavior which is decided based on a discrete uniform distribution. A trace with only one activity cannot be selected for noise insertion.

3.3.2 Simulating a Log from a Process Tree

There are many implementations for simulating business process models into event logs (e.g. [17, 35, 53, 79]). These implementations work with Coloured Petri nets, BPMN and Declare as language of the input models. However, none of the existing implementations accepts a process tree as input. Furthermore, we need the simulation to take into account the branching probabilities of the nodes in our generated trees. Using an existing simulator would require an extension to handle these probabilities. Therefore, we have opted to make a new simulation algorithm part of the "PTandLogGenerator" that accepts the generated trees with branching probabilities.

The new simulation algorithm, as described by Algorithm 2, takes a process tree, the number of traces to generate, and the noise probability as input parameters. The algorithm builds a simple event log, which is a multiset of simple traces. For each trace to be generated, it calls the function "GenerateTrace" recursively starting with the root node of the tree.

"GenerateTrace" (see Algorithm 3) takes a node and the current trace as input. At the start of the trace generation, the input node is the root node and the current trace is empty, i.e. $\sigma = \langle\rangle$. "GenerateTrace" tests the type of the input node n to decide how the current trace is expanded or which algorithms needs to be executed next. In case the input node is a leaf node (rules 7 to 10 of Algorithm 3), then the activity is added to trace, except when its label is τ (representing an invisible activity), and the updated trace is returned. In case the input node is not a leaf node but an operator node (rules 11 to 20), then, depending on the operator type, another algorithm is called with the input node

Algorithm 2 : Simulate Process Tree into event log

```
1: Input:
2:     $PT$: Process Tree
3:     $t$: the number of traces
4:     $\Pi^{Noise}$: the amount of noise
5: Output:
6:     $L$: event log
7: Start SimulateTree($PT, t, \Pi^{Noise}$)
8: for $i \in [1, t]$ do                                   ▷create t entities
9:     $\sigma_i \leftarrow \langle\rangle$                  ▷start with an empty trace
10:    $\sigma_i \leftarrow$ GenerateTrace(r,$\sigma_i$)      ▷start from the root of the tree, see Algorithm 3
11:    if $\Pi^{Noise} > 0$ then
12:        $x \leftarrow random \in [0, 1)$
13:        if $|\sigma_i| > 1$ and $x < \Pi^{Noise}$ then     ▷exclude traces of length one
14:            type ← random(head,body,tail,swap,add)
15:            add noise type to $\sigma_i$
16:        end if
17:    end if
18:    $L \leftarrow L \cup \sigma_i$                         ▷add trace to the log
19: end for
20: return $L$
```

and current trace. The following paragraphs discuss each of these algorithms.

3.3.2.1 Execute sequence operator

Algorithm 4 describes how to execute a sequence operator node. It iterates over each child node of the input sequence node and updates the current trace by a recursive call on "GenerateTrace" using the child node and current trace as inputs. To illustrate, given a node $\rightarrow \langle a, b\rangle$ and an empty trace $\sigma = \langle\rangle$, then Algorithm 4 first calls "GenerateTrace" with "a" and σ as inputs. The updated trace then contains activity "a": $\sigma = \langle a\rangle$. A similar execution happens for "b" such that Algorithm 4 in the end returns the updated trace $\sigma = \langle a, b\rangle$.

Algorithm 3 : Generate trace

```
1: Input:
2:       $n$: tree node
3:       $\sigma$: trace
4: Output:
5:       $\sigma$: updated trace
6: Start generateTrace($n,\sigma$)
7: if $n \in N_L$ and $m(n) = \tau$ then
8:     return $\sigma$
9: else if $n \in N_L$ then
10:    return $\sigma \oplus \langle m(n) \rangle$
11: else if $n \in N_\rightarrow$ then
12:    return ExecuteSequence($n,\sigma$)                    ▷Algorithm 4
13: else if $n \in N_\times$ then
14:    return ExecuteChoice($n,\sigma$)                      ▷Algorithm 5
15: else if $n \in N_\wedge$ then
16:    return ExecuteParallel($n,\sigma$)                    ▷Algorithm 6
17: else if $n \in N_\vee$ then
18:    return ExecuteOr($n,\sigma$)                          ▷Algorithm 8
19: else
20:    return ExecuteLoop($n,\sigma$)                        ▷Algorithm 7
21: end if
```

Algorithm 4 : ExecuteSequence

```
1: Input:
2:       $n$: sequence node
3:       $\sigma$: trace
4: Output:
5:       $\sigma$: updated trace
6: Start ExecuteSequence($n,\sigma$)
7: for $n_{child} \in c(n)$ do          ▷iterate over children from left to right
8:     $\sigma \leftarrow$ GenerateTrace($n_{child},\sigma$)                  ▷update trace
9: end for
10: return $\sigma$
```

3.3.2.2 Execute choice operator

Algorithm 5 describes how to execute a choice operator node. It starts by picking a random child node of the input choice node. This random selection uses the execution probabilities of each child node which can be accessed by the function $b(n_{child})$. Then, Algorithm 5 returns the updated trace by a recursive call on "GenerateTrace" using the selected child node and current trace as inputs. To illustrate, given a node $\times\langle a,b,c\rangle$ and an empty trace $\sigma = \langle\rangle$, then Algorithm 5 first randomly picks one of the three child nodes. Suppose that $b(a) = 0.9$, $b(b) = 0.05$, $b(c) = 0.05$, this means there is a 90% probability to pick "a", 5% probability to pick "b", and 5% probability to pick "c". Algorithm 5 returns the updated trace which in this small example consists of one activity, e.g. $\sigma = \langle a\rangle$.

Algorithm 5 : ExecuteChoice

```
1: Input:
2:     $n$: choice node
3:     $\sigma$: trace
4: Output:
5:     $\sigma$: updated trace
6: Start ExecuteChoice($n$,$\sigma$)
7: $n_{child} \leftarrow$ random child $\in c(n)$ based on branch probabilities
8: return $\sigma \leftarrow$ GenerateTrace($n_{child}$,$\sigma$)        ▷update trace
```

3.3.2.3 Execute parallel operator

Algorithm 6 describes how to execute a parallel operator node. It keeps track of the executed child nodes in a variable "executedChildren" which is an empty set in the beginning. While there are some child nodes that are not yet executed, the algorithm picks a random child node that has not been executed (rule 9). Then, it updates the current trace by a recursive call on "GenerateTrace" using the selected child node and current trace as inputs. After the execution of the child node, it is added to the set of executed child nodes (rule 11). Finally, after executing all of the child nodes, Algorithm 6 returns the updated trace. To illustrate, given a node $\wedge\langle a,b\rangle$ and an empty trace $\sigma = \langle\rangle$, then Algorithm 6 randomly picks one of the two child nodes. Suppose it executes "b" first, which updates the trace to $\sigma = \langle b \rangle$. As child node "b" is already executed, Algorithm 6 executes the only other child node "a", which updates the trace to $\sigma = \langle b, a \rangle$ which is returned.

Algorithm 6 : ExecuteParallel

```
1: Input:
2:     $n$: parallel node
3:     $\sigma$: trace
4: Output:
5:     $\sigma$: updated trace
6: Start ExecuteParallel($n$,$\sigma$)
7: executedChildren ← {}
8: while executedChildren $\neq \{child | child \in c(n)\}$ do    ▷execute all children in random order
9:     $n_{child}$ ← random($\{child | child \in c(n)\} \setminus$ executedChildren)
10:    $\sigma$ ← GenerateTrace($n_{child}$,$\sigma$)    ▷update trace
11:    executedChildren ← executedChildren $\cup n_{child}$
12: end while
13: return $\sigma$
```

3.3.2.4 Execute loop operator

Algorithm 7 describes how to execute a loop operator node. It always starts with executing the first child of the loop node which updates the trace by a recursive call on "GenerateTrace" using the first child node and the current trace as inputs. Then, multiple iterations of executing the second child node followed by the first child node can occur. Algorithm 7 keeps track of the number of iterations done using variable i. The maximum number of iterations k is attached to the loop operator node itself, i.e. $\circlearrowleft^k$. If the number of iterations performed i is smaller than k, then there is a 50% probability to actually do the iteration of executing the second child node followed by the first child node. Notice that i is updated no matter whether the iteration will actually occur. As a result, not all traces necessarily contain the maximum number of iterations k. When the number of iterations i equals k, the algorithm exits the while loop and executes the third child of the loop operator node and returns the updated trace (rule 16). To illustrate, given the node $\circlearrowleft^2 \langle a, b, c \rangle$ and an empty trace $\sigma = \langle\rangle$, then Algorithm 7 starts with executing the first child node which results in $\sigma = \langle a \rangle$. Suppose, that the random number is indeed smaller than 0.5 and the algorithm executes the second child followed by the first child to obtain $\sigma = \langle a,b,a \rangle$. Because $i = 1$ is smaller than k, another iteration can occur. Suppose this happens, then it results in $\sigma = \langle a,b,a,b,a \rangle$. Because $i = k$, the while loop ends and the algorithm executes the third child which updates the trace that is returned, i.e. $\sigma = \langle a,b,a,b,a,c \rangle$.

Algorithm 7 : ExecuteLoop

```
1: Input:
2:     $n$: loop node
3:     $\sigma$: trace
4: Output:
5:     $\sigma$: updated trace
6: Start ExecuteLoop($n$,$\sigma$)
7: $\sigma \leftarrow \sigma \oplus$ GenerateTrace($c(n)_1$,$\sigma$)        ▷first execute first child
8: $i = 0$
9: while $i < k$ do                                     ▷k is given for a loop node $\circlearrowleft^k$
10:    $i \leftarrow i + 1$                               ▷update the number of iterations done
11:    if randomNumber < 0.5 then
12:        $\sigma \leftarrow$ GenerateTrace($c(n)_2$,$\sigma$)           ▷execute second child
13:        $\sigma \leftarrow$ GenerateTrace($c(n)_1$,$\sigma$)           ▷execute first child
14:    end if
15: end while
16: return $\sigma \leftarrow$ GenerateTrace($c(n)_3$,$\sigma$)          ▷execute third child
```

3.3.2.5 Execute or operator

Algorithm 8 describes how to execute an “or” operator node. The algorithms starts with randomly determining the number of child nodes to be executed. The minimum number of child nodes to execute is one child node and the maximum is all the child nodes (rule 7). It keeps track of the executed child nodes in a variable “executedChildren” which is an empty set in the beginning. While the number of executed child nodes is smaller than the number of child nodes to execute, the algorithm picks a random child node that has not been executed (rule 10). Then, it updates the current trace by a recursive call on “GenerateTrace” using the selected child node and the current trace as inputs. After the execution of the selected child node, it is added to the set of executed child nodes (rule 12). Finally, after executing the selected number of the child nodes, Algorithm 8 returns the updated trace. To illustrate, given a node $\vee\langle a,b,c\rangle$ and an empty trace $\sigma = \langle\rangle$, then Algorithm 8 randomly determines the number of child nodes to execute, e.g. "numberOfChildren" = 2. As there are no child executed yet, the algorithm can randomly pick any of the three child nodes. Suppose it executes "b" first, which updates the trace to $\sigma = \langle b\rangle$. As child node "b" is already executed, Algorithm 8 executes one of the other two child nodes "a" or "c". If it picks "c", this updates the trace to $\sigma = \langle b,c\rangle$ and the determined number of executed children is reached and the trace is returned.

Algorithm 8 : ExecuteOr

```
1: Input:
2:     $n$: or node
3:     $\sigma$: trace
4: Output:
5:     $\sigma$: updated trace
6: Start ExecuteOr($n,\sigma$)
7: numberOfChildren ← random(1,$|c(n)|$)    ▷choose how many children to execute
8: executedChildren ← {}
9: while |executedChildren| < numberOfChildren do    ▷execute in random order
10:     $n_{child}$ ← random($\{child|child \in c(n)\}\setminus$ executedChildren)
11:     $\sigma$ ← GenerateTrace($n_{child},\sigma$)    ▷update trace
12:     executedChildren ← executedChildren $\cup n_{child}$
13: end while
14: return $\sigma$
```

3.3.2.6 Adding noise to a generated trace

Finally, the simulation algorithm may add noise to the generated trace (rules 11 to 17 in Algorithm 2). Noise is possibly added if the user specified probability Π^{Noise} is larger than zero. In that case, first a random number between zero and one is drawn. If the random number is smaller than the noise probability

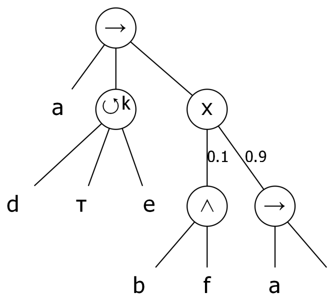


Figure 3.14: Example tree used to illustrate the simulation algorithm

Π^{Noise} and the length of the generated trace is larger than one, a random noise type is added to the trace. The trace length condition is necessary as removing an activity would produce an empty trace. Also, swapping two random activities in a trace of length one would be impossible. After the noise addition the trace is added to the log (rule 18) and new traces are generated until the specified number of traces t reached.

3.3.2.7 Example trace execution of a given process tree

To illustrate the complete simulation algorithm (Algorithm 2), consider the tree in Figure 3.14 with $k = 2$ the maximum number of loop iterations. We generate one trace which is empty in the beginning: $\sigma_1 = \langle\rangle$. The simulation starts with executing the root of the trace, i.e. the sequence node. The execution of the first child, i.e. leaf node "a", is added to the trace: $\sigma_1 = \langle a \rangle$. Then, the next child of the sequence that corresponds to a loop is executed. First, the algorithm performs the left child of the loop, i.e. leaf node "d", resulting in $\sigma_1 = \langle a, d \rangle$. Then, the algorithm randomly decides whether to exit the loop by executing the right child or repeating by performing the middle child of the loop followed by the first child again. Suppose that it decides to repeat which changes the trace to $\sigma_1 = \langle a, d, d \rangle$. Once more, the algorithm can do a repetition or exit the loop by executing the third child of the loop. Suppose, the latter is randomly chosen to change the trace in $\sigma_1 = \langle a, d, d, e \rangle$. The last part of the sequence is a choice node with unequal branching probabilities. The algorithm randomly choses the left child which results in the parallel execution of activity "b" and "f". Suppose that "f" is done first to produce the complete trace $\sigma_1 = \langle a,d,d,e,f,b\rangle$. Finally, the algorithm can inject noise into the trace. Suppose the probability to inject noise $\Pi^{Noise} = 0.1$ and a randomly drawn number x is smaller than 0.1. This means that the algorithm will inject a random type of noise, e.g. "remove head", into the trace. The removal of $\langle d,e\rangle$ ends in the noisy trace $\sigma_1 = \langle a,d,f,b\rangle$. The simulation algorithm will repeat the trace generation until it contains the number of traces specified by the user and then returns the log as input of process discovery evaluations.

The "PTandLogGenerator" presented above generates control-flow process models and event logs that only contains traces, i.e. the ordering information between activities. These logs are sufficient as input of process discovery evaluation. However, decision mining techniques require event logs that contain information on both control-flow and case/data-flow perspectives (a.k.a. the decision perspective). The next section will describe how the presented model and log generation algorithms can be extended to produce such multiperspective event logs.

3.4 Data-flow extension

Decision mining algorithms discover the routing decision logic in a process using the available case information in the event log. Therefore, an empirical evaluation of such algorithms needs process models and event logs that contain case attributes that determine the routing decision logic. We extend the GED methodology for generating control-flow models and event logs with case attributes (a decision dimension).

The first step of the GED methodology involves the definition of a model population. Definition 3.3 contained only control-flow related process patterns. The requirements in Table 2.4 specify that we need to add the determinism level dl to the model population to control the determinism of routing decisions in a process model: MP = (minimumVisibleAct, modeVisibleAct, maxVisibleAct, Π^{Base},Π^{τ},Π^{Re},Π^{Lt},Π^{In},dl). The determinism level dl is a number between 0 and 1 that specifies the average determinism of all routing decisions in a model, where 0 represents non-deterministic routing decisions, and 1 represents fully deterministic routing decisions.

The second step and third step of the GED methodology generate a sample of control-flow models and logs from a specified model population. This section will introduce the “DataExtend” that includes algorithms for adding case attributes that express the routing decision logic to process models and event logs. “DataExtend” will first draw a random process tree from a model population, analogous to ‘PTandLogGenerator’. Then, “DataExtend” will enrich the routing decisions in a process tree with case attribute information such that a routing decision depends on the value of the case attributes. Finally, “DataExtend” will simulate the process tree with case attributes into an event log for empirical decision mining evaluation.

The next part of this section presents the idea behind “DataExtend” using an example, followed by a formal description.

3.4.1 Illustration of generating multiperspective logs

The make-to-order process as shown in Figure 3.15 will be used as an example to illustrate the generation of routing decision dependencies. The process handles the production of a customer order: it starts with issuing the customer order, then materials are prepared, the products are produced, possibly followed by an inspection, then products are packaged, and finally, the products are delivered or the order is canceled when something went wrong. It contains three XOR-splits, i.e. routing decisions where choices between multiple activities need to be made:

- the first decision is whether to use new materials or mixed (recycled and new) materials,
- the second decision is about the inspection of the produced products: no inspection, a normal inspection or a thorough inspection,
- the third decision specifies whether the products will be delivered or canceled.

The process model in Figure 3.15 presents the control-flow perspective of the process, i.e. it does not contain information on the case attributes influencing the

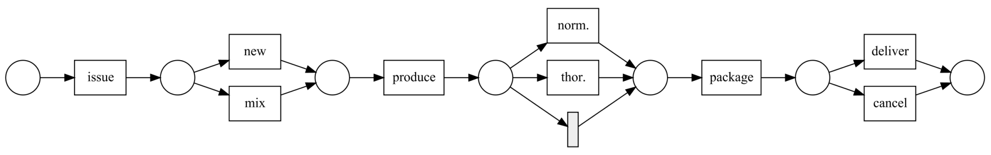


Figure 3.15: Petri net representing make-to-order example process. Activity names are abbreviated: "issue": issue order, "new": prepare new materials, "mix": prepare mixed materials, "produce": produce order, "norm.": inspect normally, "thor.": inspect thoroughly, "package": package products, "deliver": deliver products, "cancel": cancel delivery.

routing decisions. For each of the three routing decisions in the make-to-order process we can now add decision dependencies in the form of decision rules. *We assume that routing decisions can depend on order information (case attributes) or on earlier made routing decisions*.

The first routing decision regarding the use of new or mixed materials cannot depend on an earlier made routing decision in the process, but it can depend on other case attributes. However, "DataExtend" does not require each routing decision to depend on case attributes. Suppose that in this process the routing decision between new and mixed materials relies upon some contextual information not embedded in the underlying information system. Then we represent the routing decision stochastically by assigning a probability of choosing each alternative branch: there is a probability of 0.5 to execute "prepare new materials" and an equal probability to execute "prepare mixed materials".

The second routing decision about the inspection can depend on the previous routing decision and/or other case attributes. In this example the second routing decision depends on the first routing decision and a case attribute "premium" which is related to the customer placing the order. The policy is that products produced with mixed materials always need to be inspected thoroughly regardless of what the customer type is. Products consisting of new materials are only inspected for premium customers, otherwise the inspection is skipped to save costs. These routing decision dependencies can be represented as rules as illustrated in Table 3.3. A hyphen represents an indifference with regard to the value of a certain case attribute, e.g., when the first routing decision chose

	inputs		*output*
Rule number	**Routing Decision 1**	**Premium?**	**Routing Decision 2**
1	prepare new	True	inspect normally
2	prepare new	False	(skip inspection)
3	prepare mix	-	inspect thorougly

Table 3.3: Decision table for second routing decision

	input	*output*
Rule number	**Acceptable Quality?**	**Routing decision 3**
1	True	deliver
2	False	cancel

Table 3.4: Decision table for third routing decision

"prepare mix", the value of "premium" does not matter as it always leads to "inspect thoroughly".

The third routing decision regards the delivery of the produced products. The routing decision could depend on the outcome of the first and second routing decision and/or some other case attribute(s). Suppose that the inspection in the second routing decision results in an outcome indicating that the quality of the products is acceptable or non-acceptable. If an inspection was skipped, acceptable quality of the products is assumed. A delivery will only be executed if the quality of the products are acceptable, otherwise the order is cancelled. These routing decision dependencies can be illustrated as shown in Table 3.4.

The model of the produce order process together with the above routing decision dependencies can then be simulated into an event log. The simulator evaluates the rules tied to each routing decision in the process in order to decide which alternative branch to execute. An example case is shown in Table 3.5. The process starts with activity "issue order", followed by the first routing decision which is stochastic. Suppose that activity "prepare mix" is chosen randomly. Then, the order is produced (activity "produce") followed by a routing decision that takes into account the rules in Table 3.3 to decide on the inspection. Activity

Event ID	Activity	Routing Decision 1	Acceptable Quality?	Premium?
1	issue			True
2	prepare mix	prepare mix		True
3	produce	prepare mix		True
4	inspect thoroughly	prepare mix	True	True
5	package	prepare mix	True	True
6	deliver	prepare mix	True	True

Table 3.5: Example trace of the make-to-order process with case/data perspective.

"inspect thoroughly" is executed because the first routing decision chose "prepare mix" and the value of the attribute "premium" is "true". The inspection produces the "true" value for the case attribute "Acceptable quality". Finally, the products are packaged (activity "package") and the first rule corresponding to the last routing decision is valid, resulting in the execution of activity "deliver".

Link with DMN

For the reader familiar with BPMN [46] and DMN [47], the example is translated to those formalisms to avoid confusion of the used terminology. BPMN with DMN offers the option to integrate the control-flow with the decision layer of a process. As such they offer an alternative representation of the integration in this thesis.

"DataExtend" adds routing decision dependencies to each routing decision. Those dependencies consist of case attributes and previous routing decisions. In BPMN-DMN terminology this means that we assume each routing decision to be preceded by a decision that influences the outgoing branch chosen in the routing decision. This is visualized by the business rule tasks, i.e. "Determine materials" (routing decision 1), "Decide inspection" (routing decision 2), and "Decide delivery" (routing decision 3) in Figure 3.16 that represents the same make-to-order process as the model in Figure 3.15. The dependencies of a routing decision on previous routing decisions and case attributes can be represented using the Decision Requirements Diagram (DRD) of DMN, i.e. each of the decisions in the DRD corresponds to one business rule task in the process model in Figure 3.16. The two DRD's in Figure 3.17 show the dependencies of "Decide inspection" (routing decision 2) and "Decide delivery" (routing decision 3). For each decision in the DRD, "DataExtend" generates the decision logic in a decision table similar to a DMN decision table with a collect hit policy, i.e. when multiple rules hold then a random rule is chosen from those.

The next subsection formalizes the steps taken in this example for the "DataExtend" approach of generating event logs with case attributes.

3.4.2 Formal steps to generate multiperspective logs

"DataExtend" extends the previously proposed algorithms for model and log generation (see Section 3.2 and 3.3) with case attributes.

3.4.2.1 Extending routing decisions with case attributes

"DataExtend" starts from a pure control-flow process model drawn from a specified model population. To be consistent with the model generation, process trees are used to represent these models. The first assumption of "DataExtend" is that routing decisions correspond only to choice operators in a tree. As such, we do not consider case attributes on routing decisions part of loop and "or" operators. Secondly, "DataExtend" assumes that routing decisions can depend on case attributes including previous routing decisions. Notice that an execution of a routing decision creates a case attribute that contains the chosen branch. This means that LT dependencies as described in Section 3.2.4 can be added using case attributes. Based on these assumptions, "DataExtend" requires process trees without LT dependencies as input, i.e. no unfolded choice trees. The following steps can be added to the end of the tree building algorithm shown in Figure 3.4.

"DataExtend" starts from the set of all choice operator nodes $N_\times$ in a given process tree as routing decisions. Then, it assigns zero or more case attributes randomly to each routing decision.

Definition 3.6 (Assign)**.** Given a set of routing decisions $N_\times$ and a set V of case attributes (including previous routing decisions), Assign: $N_\times \mapsto \mathbb{P}(V)$ is a

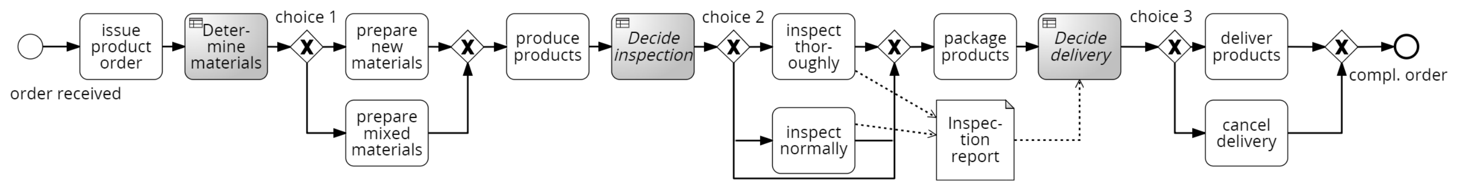


Figure 3.16: Make-to-order process in BPMN notation with business rule tasks and data objects.

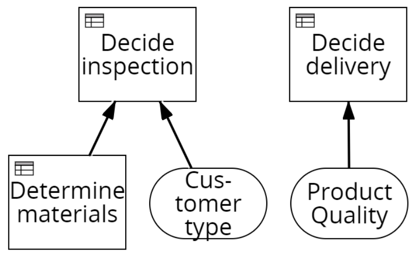


Figure 3.17: Decision requirements diagrams for the make-to-order process.

function that labels each routing decision $\times_i$ with a set $V' \subseteq V$ of attributes which $\times_i$ is based upon.

The assigned case attributes of a routing decision can include the previous routing decisions. These previous decisions can be identified using the precedence function:

Definition 3.7 (Precedence)**.** Precedence: $N_\times \mapsto \mathbb{P}(N_\times)$ is a function that labels each routing decision with a set of preceding routing decisions. The precedence is based on the control-flow semantics of the model[7].

Consider again the example about the make-to-order process described in the previous subsection (see Section 3.4.1). We have translated the Petri net to a trace equivalent process tree shown in Figure 3.18. Each of the choice operators representing a routing decision has been given an index so that it can be easily referred to. In the example, the second routing decision ($\times_2$) is preceded by the first routing decision ($\times_1$): starting from $\times_2$ and working towards the root of the tree, the first common ancestor of $\times_1$ and $\times_2$ is the sequence operator $\rightarrow$ and $\times_1$ appears before $\times_2$ in the subtree of $\rightarrow$ such that Precedence($\times_2$)$\mapsto \{\times_1\}$. Then, in the example, the second routing decision is assigned the first routing decision and "premium" as case attributes: Assign($\times_2$) $\mapsto \{\times_1, premium\}$.

In a next step, "DataExtend" uses the assigned case attributes to specify how they influence each routing decision. More specifically, a routing decision corresponds to a choice between multiple alternative process paths. The values of the assigned case attributes can restrict such a choice. These restrictions, also called routing decision dependencies, can be expressed as decision rules. A decision rule is defined as a mapping:

Definition 3.8 (Decision Rule)**.** A decision rule is a mapping

$$V_1 \bowtie q_1, \ldots, V_w \bowtie q_w \mapsto \times_{jk}$$

[7]A node n_1 precedes another node n_2 in a process tree if the lowest common ancestor of the two nodes is a sequence operator $\rightarrow$ such that n_1 appears before n_2 in the sequence of nodes $s(\rightarrow)$ returned by the subtree function (see Definition 3.2) and $n_1 \neq n_2$. The lowest common ancestor of two nodes is the common ancestor that is located farthest from the root of the tree [12].

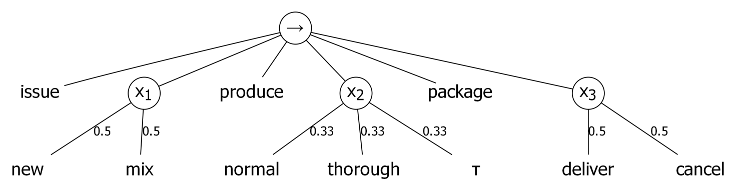


Figure 3.18: Process tree representing make-to-order example process. Activity names are abbreviated: “issue”: issue order, “new”: prepare new materials, “mix”: prepare mixed materials, “produce”: produce order, “normal”: inspect normally, “thorough”: inspect thoroughly, “package”: package products, “deliver”: deliver products, “cancel”: cancel delivery.

where $V_i \in V$ is the set of case attributes, $\bowtie$ is a relational operator $\in \{<, \leq, >, \geq, =, \neq\}$, $q_1, \ldots, q_w$ are constants, $\times_{jk}$ branch k of routing decision j, i.e. $\times_{jk} \in c(\times_j)\ and\ \times_j \in N_\times\ and\ j,k \in \mathbb{N}^+$.

The set of all decision rules related to a routing decision can be represented as a decision table such as Table 3.3, where row 2 expresses the decision rule: $\times_1$ = prepare new materials, premium = True $\mapsto$ $\times_{22}$(= inspect normally).

“DataExtend” initially makes all routing decisions free-choice by generating all possible decision rules, i.e. each possible combination of case attribute values can lead to any of the outgoing branches.[8] For example, consider Table 3.6 that shows the initial set of decision rules for routing decision 2 in the make-to-order example process. When a case has the following attribute values: routing decision 1 = “prepare new” and premium = “True”, then activities “inspect thoroughly”, “inspect normally”, and skip inspection are all possible according to rules 1, 2 and 3.

Randomly removing rules from the initial set of decision rules restricts the possible outgoing branches at each routing decision. In this way, “DataExtend” creates routing decision dependencies. However, it cannot restrict the behavior too much as this could create deadlocks and dead parts and thus violate the

[8]Impossible combinations happen when a routing decision depends on two other routing decisions that are mutually exclusive. For example in a tree $s(PT) = \rightarrow \langle \times_1 \langle \times_2 \langle a,b \rangle, \times_3 \langle c,d \rangle \rangle, e, \times_4 \langle f,g \rangle \rangle$, $\times_4$ may depend on both $\times_2$ and $\times_3$. However, $\times_2$ and $\times_3$ are mutually exclusive, e.g. a rule $\times_2 = a,\ \times_3 = c \mapsto \times_4 = f$ is impossible as “a” and “c” can never happen together. Such combinations are removed from the decision table.

	inputs		*output*
Rule number	**Routing Decision 1**	**Premium?**	**Routing Decision 2**
1	prepare new	True	inspect thorougly
2	prepare new	True	(skip inspection)
3	prepare new	True	inspect normally
4	prepare new	False	inspect thorougly
5	prepare new	False	(skip inspection)
6	prepare new	False	inspect normally
7	prepare mix	True	inspect thorougly
8	prepare mix	True	(skip inspection)
9	prepare mix	True	inspect normally
10	prepare mix	False	inspect thorougly
11	prepare mix	False	(skip inspection)
12	prepare mix	False	inspect normally

Table 3.6: Example initial decision table for the second routing decision in the make-to-order example process.

soundness requirement. The process tree extended with routing decision rules is in fact a decision-aware process model. Therefore, the definition of soundness of decision-aware proces models by Batoulis and Weske [9] can be adopted here: a decision-aware process model is sound iff: the model is (classical) sound, it is decision-deadlock free and contains no dead decision branches. The first requirement of classical soundness is guarantueed by process trees that are inherently sound. To guarantee the last two requirements, the following soundness constraints similar to the constraints introduced in [9] are imposed on the rule removal step:

- each set of decision rules, i.e. all decision rules related to one routing decision (illustrated as a decision table), has at least one rule for each possible routing outcome to prevent *dead activities*

- each set of decision rules has at least one rule for each value combination of case attributes values to prevent *deadlocks*

More recent literature [8, 26] on decision-aware soundness discovered that these constraints are necessary but not sufficient to guarantee soundness. The reason the adopted soundness constraints are insufficient is because they only check the local interplay of routing decision rules and routing decisions in the process model. The local interplay does not take into account that routing decisions may depend on other routing decisions. In that case deadlocks or dead activities may still occur. This is a threat to the validity of "DataExtend" that should be researched further. Section 3.4.2.2 describes a workaround in case of unsoundness.

Additionally, the user can set a stopping criterion for the removal of random decision rules. Without such a stopping criterion, "DataExtend" will remove the rules until no removal can happen without violating the soundness constraints. This results in fully deterministic routing decisions, i.e. for any combination of case attribute values there is only one outgoing branch possible. However, the determinism requirement stated that a user should be able to control whether or not a routing decision is fully deterministic or non-deterministic. Therefore, the approach allows users to set a determinism level as stopping criterion. The determinism level is defined as the number of decision rules removed relative to the maximum amount of decision rules that could possibly be removed (without violating the soundness constraints). The maximum determinism level of 1 results in a fully deterministic routing decision. The minimum value of 0 denotes the initial state, i.e. a free-choice routing decision. The user specifies the target determinism level, which is the average determinism level over all routing decisions *with* case attributes after the removal of rules. We explicitly leave out routing decisions without assigned case attributes[9] because these routing decisions always have a determinism level of 0, i.e. no rules are removed which corresponds to a free-choice routing decision. Including such free-choice routing decisions makes it impossible to reach a target determinism level of 1.

Definition 3.9 (Determinism level)**.**

$$DeterminismLevel(\times_i) = \frac{initial\# - current\#}{initial\# - minimum\#}$$

[9]Recall that "DataExtend" assigns *zero* or more case attributes to a routing decision.

The numerator indicates the number of removed rules and the denominator indicates the maximum number of rules that could be removed without violating the soundness constraints. With *initial#* and *current#* indicating the initial and current number of rules related to $\times_i$, *minimum#* expresses the minimum of rules that need to remain according to the soundness constraints: at least one rule for each possible routing outcome and at least one rule for each unique combination of case attribute values.

Definition 3.10 (Average determinism level)**.** Let $N_\times^a$ be the set of routing decision nodes that have at least one or more case attributes assigned to, i.e. $N_\times^a = N_\times \setminus \{\times_j \in N_\times | Assign(\times_j) \mapsto \emptyset\}$, then the average determinism level is defined as:

$$AverageDeterminismLevel(PT) = \frac{\sum\limits_{\times \in N_\times^a} DeterminismLevel(\times)}{|N_\times^a|}$$

In the make-to-order example (see Section 3.4.1) the desired determinism level is set to 100%. This means that as much rules as possible have to be removed from the decision table related to each routing decision with case attributes, i.e. the second and third routing decision. The initial decision table for the second routing decision (see Table 3.6) contains 12 rules, i.e. *initial#* = *current#* = 12. The soundness constraints imply that at least one rule for each of the three possible decision outcomes should remain to avoid dead activities, i.e. one rule for each of the three activities: “inspect normally”, “inspect thoroughly”, and skipping the inspection. Additionally, the soundness constraints require that the decision table should contain at least one rule for each of the four unique combination of case attribute values: {“prepare new”, “true”}, {“prepare mix”, “true” }, {“prepare new”, “false”}, and {“prepare mix”, “false”}. Therefore, the minimum number of rules to remain in the decision table is the maximum of the number of rules needed to fulfill each constraint: *minimum#* = *max*(3,4) = 4. As such, the determinism level of the second routing decision *DeterminismLevel*($\times_i$) would reach a maximum of 1 when 8 rules are removed from the corresponding decision table. In the make-to-order example rules 1, 2, 4, 6, 8, 9, 11 and 12 are removed from Table 3.6. This results in

	inputs		*output*
Rule number	**Routing Decision 1**	**Premium?**	**Routing Decision 2**
1	prepare new	True	inspect normally
2	prepare new	False	(skip inspection)
3	prepare mix	True	inspect thorougly
4	prepare mix	False	inspect thorougly

Table 3.7: Decision table for second routing decision

	input	*output*
Rule number	**Acceptable Quality?**	**Routing decision 3**
1	True	deliver
2	False	cancel

Table 3.8: Decision table for third routing decision

the decision Table 3.7 with a determinism level: $\frac{12-4}{12-max(3,4)} = 1$. Similarly, decision rules are removed for routing decision three that ends in the decision Table 3.8 with determinism level 1. This makes the average determinism level $\frac{DeterminismLevel(\times_2)+DeterminismLevel(\times_3)}{|\{\times_2,\times_3\}|} = \frac{1+1}{2}$ equal to 1 as all routing decisions with case attributes are fully deterministic.

Algorithm 9 summarizes the steps presented above to extend routing decisions with data dependencies.

3.4.2.2 Simulating routing decisions with case attributes

After adding routing decision dependencies (decision rules) to the choices in a process tree, “DataExtend” will simulate those trees with rules into an event log. Different from the simulation algorithm that generated a simple event log (see Algorithm 2), “DataExtend” creates a “rich” event log as specified in Definition 2.2. It takes the number of cases to be generated, the process tree, and the set of routing decision rules as input. Then, at the start of each case,

Algorithm 9 : Extend process tree with routing decision logic

```
1: Input:
2:     $PT$: process tree
3:     $dl$: target determinism level
4: Output:
5:     $PT$: process tree
6:     $\mathscr{R}$: set of routing decision rules
7: Start ExtendTree($PT, dl$)
8: for $\times_i \in N_\times$ do
9:     Assign($\times_i$) $\mapsto V_{random}$                ▷Assign random case attributes
10:    $R_{\times_i} \leftarrow$ enumerateCombinations(Assign($\times_i$)) ▷Make initial set of decision rules
11:    $\mathscr{R} \leftarrow \mathscr{R} \cup R_{\times_i}$                ▷Add to set of all rules
12: end for
13: while $AverageDeterminismLevel(PT) < dl$ do    ▷while average determinism level of routing decisions with case attributes is lower than the target determinism level
14:    Remove random rule from $\mathscr{R}$ without violating soundness constraints
15: end while
16: return $\mathscr{R}$
```

all case attributes, except the ones that correspond to a routing decision[10], are assigned a random value in their domain which remains constant during the execution of the trace. For example, in the make-to-order process the "premium" case attribute gets value "true": $\#_{premium}(c)$ =true.

The next steps of the log generation correspond to the original simulation algorithm (see Algorithm 2), i.e. a trace is generated for each case. The only changes are with regard to the "ExecuteChoice" algorithm (see Algorithm 5 above) which results in the adapted algorithm "ExecuteChoiceData" (see Algorithm 10). The choice of a random child of a routing decision $\times_i$ is possibly restricted by the generated decision rules $R_{\times_i} \in \mathscr{R}$. Therefore, "DataExtend" will collect the values of each of the assigned case attributes $\{V_1, \ldots, V_w\}$ to make a state: $V_1 = \#_{V_1}(c), \ldots, V_w = \#_{V_w}(c)$. Then it will iterate over all the decision rules to collect the possible routing decision branches. A routing decision branch is

[10]These case attributes are assigned the chosen branch as value when the corresponding routing decision is executed.

possible if a rule condition matches with the state. Finally, one of the possible branches is executed, the case attribute associated to that routing decision gets the chosen branch as value, and the trace generation continues.

Algorithm 10 : ExecuteChoiceData

```
1:  Input:
2:        $n$: choice node
3:        $c$: case
4:        $R_n$: set of routing decision rules
5:  Output:
6:        $c$: updated case
7:  Start ExecuteChoiceData($n,c,R_n$)
8:  $s \leftarrow \{V_i = \#_{V_i}(c) | V_i \in Assign(n)\}$        ▷get current state of case attributes
9:  PossibleBranches ← {}
10: for $rule \in R_n$ do
11:     if condition of rule coincides with $s$ then
12:         PossibleBranches ← PossibleBranches ∪ branch mapped to condition
13:     end if
14: end for
15: $n_{child}$ ← random child ∈ PossibleBranches based on branch probabilities
16: $\#_{trace}(c) \leftarrow$ GenerateTrace($n_{child}, \#_{trace}(c)$)        ▷update trace
17: return $c$
```

As described in Section 3.4.2.1, the soundness constraints can lead to deadlocks and dead activities in case a routing decision depends on another routing decision. Dead activities do not cause errors in the simulation as they may only lead to some routing decision outcomes that are never executed. To the contrary, deadlocks may interrupt the simulator. To deal with this, the simulator makes a workaround: if the state of a case at a routing decision does not coincide with any of the rules related to that routing decision, then the simulator executes a random branch of the routing decision.

The simulation of a process tree with routing decision logic yields an event log with both control-flow and case information as needed for decision mining evaluation.

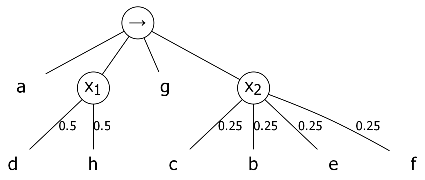


Figure 3.19: Original process tree with two routing decisions $\times_1$ and $\times_2$

3.4.3 Illustration of automatic routing decision logic generation

This subsection will provide another example to illustrate how the above method for extending routing decisions in process trees with case attributes is automated.

Consider the original process tree shown in Figure 3.19. The tree has two routing decisions $\times_1$ and $\times_2$. Before the routing decisions can be extended with case attributes, a few parameters need to be set by the user. The first parameter is the target determinism level of all routing decisions which is set to 0.5, i.e. the routing decisions should be non-deterministic. The suggested implementation of "DataExtend" can handle three types of case attributes: Boolean, string, and numerical attributes. To ensure that there is finite number of possible routing decision rules, the numerical attributes are discretized with intervals. As such, the second parameter a user needs to specify is the uniform distribution of the number of intervals $\#i$, e.g. $\#i \sim$ uniform $(2,3)$, i.e. each numerical attribute is discretized to minimum two and maximum three intervals. The third parameter involves the number of case attributes $\#c$ assigned to each routing decision. For example, $\#c \sim$ uniform $(0,2)$, i.e. each routing decision is assigned minimum zero and maximum two case attributes.

In the first step of the data extension, Algorithm 9 iterates over all routing decisions in the given process tree. A random number from $\#c \sim$ uniform $(0,2)$ is drawn to determine the number of case attributes for routing decision 1. Suppose that $\#c = 1$ such that routing decision 1 gets one case attribute. As there are no case attributes assigned before and there is no preceding routing decision (Precedence($\times_1$) = {}), a new case attribute V_1 is created and assigned:

Rule number	*input* **V₁**	*output* **Routing decision 1**
1	[0,0.757)	d
2	[0,0.757)	h
3	[0.757,0.931)	d
4	[0.757,0.931)	h
5	[0.931,1)	d
6	[0.931,1)	h

Table 3.9: Initial decision table for first routing decision

Assign($\times_1$) $\mapsto \{V_1\}$. V_1 gets a random type, for example numerical. Then, the number of intervals is randomly drawn from $\#i \sim$ uniform $(2,3)$. Suppose that $\#i = 3$ such that V_1 has a domain $[0,1)$ and two randomly chosen cutoff points determine that the routing decision rules can use three intervals for V_1, e.g. $[0,0.757)$, $[0.757,0.931)$ and $[0.931,1)$. Note that interval $[0,0.757)$ is equivalent to $0 \leq V_1 < 0.757$. Similar to routing decision 1, a random number from $\#c \sim$ uniform $(0,2)$ is drawn to determine the number of case attributes for routing decision 2. Suppose that $\#c = 2$ such that routing decision 2 will have two case attributes assigned. If there is a preceding routing decision, in this case there is as Precedence($\times_2$) $= \{\times_1\}$, then this routing decision is assigned first. If there are already other case attribute assigned, then there is a 50% probability that they are assigned again and a 50% probability a new case attribute is created and assigned. Suppose that the second routing decision is assigned the first routing decision and a new case attribute V_2 which is of type Boolean: Assign($\times_2$) $\mapsto \{\times_1, V_2\}$.

Following the assignment of case attributes to the routing decisions, the initial sets of rules are created. Take all the combinations of possible values of V_1 and outgoing branches of routing decision 1 to create $R_{\times_1}$, i.e. $\{[0,0.757)$, $[0.757,0.931)$, $[0.931,1)\} \times \{d,h\}$ which results in the rules in Table 3.9. Take all the combinations of possible values of V_2, outgoing branches of routing decision 1 and outgoing branches of routing decision 2 to create $R_{\times_2}$, i.e. { True, False } $\times$ $\{d,h\} \times \{c,b,e,f\}$ which results in the rules in Table 3.10.

The average determinism level of the original tree with the created initial

		inputs	*output*
Rule number	**V_2**	**Routing decision 1**	**Routing decision 2**
1	True	d	c
2	True	d	b
3	True	d	e
4	True	d	f
5	True	h	c
6	True	h	b
7	True	h	e
8	True	h	f
9	False	d	c
10	False	d	b
11	False	d	e
12	False	d	f
13	False	h	c
14	False	h	b
15	False	h	e
16	False	h	f

Table 3.10: Initial decision table for second routing decision

routing decision rules is 0 as both routing decision rules are free-choice. To reach the target determinism level of 0.5, rules are randomly removed from any of the decision tables, except when the removal violates the soundness constraints. Removing rule 7 in Table 3.10 increases the determinism level and average determinism level:

$$DeterminismLevel(\times_2) = \frac{16-15}{16-4} = \frac{1}{12}$$

$$AverageDeterminismLevel(PT) = \frac{\frac{1}{12}+0}{2} = \frac{1}{24}$$

The removal of rules continues until the target determinism level is reached. Suppose that rules 2 and 3 are removed from Table 3.9 and rules 7, 10, 12, and 16 are removed from Table 3.10 which result in Tables 3.11 and 3.12 with:

$$AverageDeterminismLevel(PT) = \frac{\frac{6-4}{6-3}+\frac{16-12}{16-4}}{2} = 0.5$$

	input	*output*
Rule number	$\mathbf{V_1}$	**Routing decision 1**
1	$[0, 0.757)$	d
4	$[0.757, 0.931)$	h
5	$[0.931, 1)$	d
6	$[0.931, 1)$	h

Table 3.11: Final decision table for first routing decision

		inputs	*output*
Rule number	$\mathbf{V_2}$	**Routing decision 1**	**Routing decision 2**
1	True	d	c
2	True	d	b
3	True	d	e
4	True	d	f
5	True	h	c
6	True	h	b
8	True	h	f
9	False	d	c
11	False	d	e
13	False	h	c
14	False	h	b
15	False	h	e

Table 3.12: Final decision table for second routing decision

Finally, the final sets of routing decision rules with the original process tree are simulated into an event log. Each case generation starts with drawing random values for the case attributes V_1 and V_2 from their respective domains, i.e. $[0, 1)$ and { True, False }. Suppose that $V_1 = 0.22$ and $V_2 =$ True. Then the simulation starts with executing activity "a" which is followed by the first routing decision. Because $V_1 = 0.22$, rule 1 in Table 3.11 is the only rule that holds and activity "d" is executed. This updates the value of case attribute: $\times_1 =$ "d". Next, the case generator executes activity "g" and reaches the second routing decision. As $V_2 =$ True and $\times_1 =$ "d", rules 1, 2, 3, and 4 of Table 3.12 hold which means that a random branch from the second routing decision is chosen, e.g. activity “e” is executed. This results in the following trace of newly generated case c: $\#_{trace}(c) = \langle a, d, g, e \rangle$. The case generation continues until the number of cases equals the amount specified by the user.

3.5 Demonstration and evaluation

The previous sections discussed the design and development of the GED methodology, the ‘PTandLogGenerator’ and the “DataExtend” method. As required in Design Science research, this section will discuss the demonstration and evaluation of the developed artefacts.

3.5.1 Tool implementation

Empirical analysis of process discovery and decision mining algorithms typically requires an extensive set of experiments. Therefore, the “PTandLogGenerator” and “DataExtend” should be automated for its application in empirical analysis. At the same time, the automation needs to comply to the third group of requirements of GED, i.e. standard formats and integration within the ProM Framework [120] (see Table 2.4). For the standard formats, this means that the output process trees and event logs should be in the PTML and XES standard formats [1] respectively.

Two tool implementations are available: one Python package and one package with plugins in the ProM framework. The Python package is available on Github.[11] The package contains programs callable from command line for generating random process trees, generating event logs from those trees, and generating multiperspective event logs from process trees using “DataExtend”. The ProM package *PTandLogGenerator* [59] includes the plugins “Generate Process Trees from Population”, “Generate Log Collection (with noise) from Process Trees”, and “Generate Data Log Collection from Process Trees”. Each of the tools support the necessary standard formats.

[11] `https://github.com/tjouck/PTandLogGenerator`

3.5.2 Data generation setup

To demonstrate the parameter setup of the "PTandLogGenerator" and "DataExtend", two use cases are designed. The first use case evaluates the performances of a set of process discovery algorithms. In such a case, the evaluation requires multiple process models and event logs with an extensive set of control-flow patterns to avoid an oversimplified evaluation. The second use case evaluates decision mining algorithms on event logs with non-deterministic routing decisions. The first use case requires the following three steps.

In the first step, the model populations are defined (see Table 3.13). The definition of a model population MP (see Definition 3.3) requires the specification of the top 12 parameters in column 1 of Table 3.13. This demonstration uses two model populations MP_{New} and $\text{MP}_{\text{Existing}}$ each with different parameter settings, except for the number of visible activities which varies between 10 and 30. The MP_{New} population contains all the base patterns, silent and reoccurring activities and choices with infrequent paths. Additionally it contains LT dependencies for which loops with choices are unfolded with a maximum of one repetition. The $\text{MP}_{\text{Existing}}$ population contains all the patterns available in current state-of-the-art tool PLG2 [17]. Therefore, $\text{MP}_{\text{Existing}}$ does not contain "or", reoccurring activities and LT dependencies.

In the second step, a sample of models is drawn from each model population (see Table 3.13). Finally, in the third step, the simulator will generate event logs from the trees in the sample. The simulation parameters to set are the number of logs per tree, the number of traces in the log and the probability of noise insertion (see Table 3.13). For the two model populations, the demonstration will generate one event log per tree, each log containing 1000 traces and 10% noise probability.

The second use case requires a user to define a model population and extend it with a target determinism level value. Recall that the model population cannot contain LT dependencies, therefore, we will use the control-flow parameters settings of the $\text{MP}_{\text{Existing}}$ population and extend it with a determinism level parameter equal to 0.5 to define the MP_{Data} population (see the fourth column of Table 3.13). This parameter setting refines the routing decisions of the processes in the population as non-deterministic. Furthermore, the parameter settings for the log generation remains the same except that the "number of traces (t)" parameter now represents the number of cases which all have a trace attribute and possibly other case attributes.

The setup of the parameters as in Table 3.13 can serve as a template for future users of the GED methodology and "DataExtend" in empirical process discovery and decision mining analysis. Including this table in the report of such an analysis will clearly describe the model populations from which the samples of models and logs are drawn that are used in the experiments. Furthermore, such a table also enhances transparency and reproducibility of the experiment results.

The parameter setup of "PTandLogGenerator" and "DataExtend" have been demonstrated. The following section will focus on the evaluation of these artefacts.

3.5.3 Evaluation

The first part of the evaluation investigates whether "PTandLogGenerator" and "DataExtend" meet all the requirements stated in Table 2.4. The second part of the evaluation assesses the scalability of the "PTandLogGenerator". Finally, an empirical evaluation of four process discovery techniques validates the effectiveness of the "PTandLogGenerator".

3.5.3.1 Requirements

The *full control* requirements of GED imply that a user can control the control-flow behavior in the generated process trees (control-flow patterns) and event logs (log characteristics). Therefore, this part of the evaluation checks whether the characteristics of the sample of trees and logs of the use case are conform with the input parameters of population MP_{New} in Table 3.13. Table 3.14 displays the descriptive statistics of the tree and log sample characteristics drawn from population MP_{New}.

Firstly, the distribution of the number of visible activities conforms to the triangular distribution characterized in Table 3.13. Secondly, the mean relative

Parameter	**Population $\mathbf{MP_{new}}$**	**Population $\mathbf{MP_{existing}}$**	**Population $\mathbf{MP_{data}}$**	**Population $\mathbf{MP_{scalability}}$**
Number of visible activities	(10,20,30)	(10,20,30)	(10,20,30)	(10,20,30)
Sequence ($\Pi^{\rightarrow}$)	0.5	0.5263158	0.5263158	$\in$[0,1]
Parallel ($\Pi^{\wedge}$)	0.15	0.1578947	0.1578947	$\in$[0,1]
Choice ($\Pi^{\times}$)	0.25	0.2631579	0.2631579	$\in$[0,1]
Loop ($\Pi^{\circlearrowleft}$)	0.05	0.0526316	0.0526316	$\in$[0,1]
Or ($\Pi^{\vee}$)	0.05	0.0	0.0	$\in$[0,1]
Silent activities (Π^{τ})	0.1	0.1	0.1	0.1
Reoccurring activities (Π^{Re})	0.1	0.0	0.0	0.1
Long-term dependencies (Π^{Lt})	0.5	0.0	0.0	$\in$[0,1]
Unfold loops	True	n/a	n/a	$\in${False,True}
Max repeat (k)	1	n/a	n/a	$\in${0,1,2}
Infrequent paths (Π^{In})	0.5	0.5	0.5	0.5
Sample size (number of trees)	2000	50	2000	1000
Logs per model	1	1	1	n/a
Number of traces (t)	1000	1000	1000	n/a
Noise (Π^{Noise})	0.1	0.1	0.0	n/a
Determinism level	n/a	n/a	0.5	n/a

Table 3.13: Input Parameters of Data Generation

frequencies and the confidence intervals for these means of all the control-flow constructs are shown in the second and fourth column of Table 3.14.[12] The population values of most parameters are contained in the confidence interval of the mean and some, i.e. "choice", "loop", and "infrequent paths", only differ slightly from the interval. A noticeable exception is the confidence interval for LT dependencies, which is more than 10 percent points lower than the population value. This was caused by the pruning mechanism which prevents inserting LT dependencies that cause dead activities. As such the average percentage of LT dependencies a tree will mostly be below the population value. This means that the population value of LT dependencies should be interpreted as an upper bound for the average percentage of LT dependencies in the model sample.

The number of traces in the generated event logs are exactly as specified in the input parameters. The average percentage of noisy traces is slightly lower than the probability set in Table 3.13. This percentage was influenced by not considering traces with only one activity which has led to fewer than 1000 candidate traces in some logs. Similar to the population value of LT dependencies, the population value of noise should be viewed as an upper bound to the average percentage of noisy traces in the log sample.

Overall, the "PTandLogGenerator" satisfies the full control requirement as it effectively allows users to control the characteristics of the generated models and logs through a population, given the probability of LT dependencies and noise are considered as upper bounds. Note that the soundness requirement was already proven by Theorem 3.1 in Section 3.2.4.

Next, to the input parameters, Table 3.14 shows the mean tree and log generation time in seconds. These performances were accomplished on a laptop with an Intel Core i5-4200U processor and 8 GB of RAM memory.

The *determinism* requirement of "DataExtend" is evaluated in a similar way. We check whether the determinism of the sample of trees conforms with the determinism specified in the MP_{Data} population in Table 3.13. Table 3.15 displays the descriptive statistics of the tree and log sample characteristics drawn from population MP_{Data}. The population value of 0.5 for the determinism

[12]These mean relative frequencies of the operator types were calculated before the trees were reduced.

Parameter	**Sample mean**	**Population value**	**Confidence Interval**
Number of visible activities	(11,21,30)	(10,20,30)	/
Sequence ($\Pi^{\rightarrow}$)	0.4982	0.5	🟩 [0.4931, 0.5032]
Parallel ($\Pi^{\wedge}$)	0.1506	0.15	🟩 [0.1470, 0.1542]
Choice ($\Pi^{\times}$)	0.2544	0.25	🟥 [0.2502, 0.2586]
Loop ($\Pi^{\circlearrowleft}$)	0.0471	0.05	🟥 [0.0449, 0.0493]
Or ($\Pi^{\vee}$)	0.0498	0.05	🟩 [0.0475, 0.0520]
Silent activities (Π^{τ})	0.0976	0.10	🟩 [0.0921, 0.1032]
Reoccurring activities (Π^{Re})	0.0987	0.10	🟩 [0.0958, 0.1016]
Long-term dependencies (Π^{Lt})	0.3835	0.5	🟥 [0.3753, 0.3917]
Infrequent paths (Π^{In})	0.4833	0.5	🟥 [0.4708, 0.4958]
Mean percentage of noisy traces	0.0934	0.10	🟥 [0.0924, 0.0943]
Tree generation time (seconds)	17.849	/	[2.9137, 32.784]
Log generation time (seconds)	1.37	/	[1.3421, 1.3894]

Table 3.14: Descriptive Statistics of a Sample from Population MP_{New}. Legend for the colors of the cells that describe the confidence intervals (CI): green if CI contains the population value, red if CI does not contain population value, and no color if there is no corresponding population value.

level is not contained in the confidence interval shown in the fourth column. The determinism level of the trees in the sample is slightly higher than the population value. This is because the population value serves as a lower bound: keep removing random decision rules as long as the average determinism level is lower than the population value. The descriptive statistics for the parameters illustrate again that a user has full control over the characteristics of the generated models and logs through a population definition. Notice that in this sample more population values are contained in the confidence interval of the mean compared to Table 3.14. This is due to the random sampling. In general, the larger the sample, the more population values are contained in the confidence interval of the mean. Adding more trees to the sample of MP_{New} would result in more population values included in the confidence intervals of the sample means. Soundness is another requirement of "DataExtend". The soundness constraints introduced by "DataExtend" are not sufficient (see Section 3.4.2.1) and should be

Parameter	**Sample mean**	**Population value**	**Confidence Interval**
Determinism level	0.5250	0.5	[0.5233, 0.5266]
Number of visible activities	(11,20,30)	(10,20,30)	/
Sequence ($\Pi^{\rightarrow}$)	0.5257	0.5263	[0.5205, 0.5308]
Parallel ($\Pi^{\wedge}$)	0.1576	0.1579	[0.1539, 0.1613]
Choice ($\Pi^{\times}$)	0.2645	0.2632	[0.2599, 0.2690]
Loop ($\Pi^{\circlearrowleft}$)	0.0523	0.0526	[0.0500, 0.0546]
Silent activities (Π^{τ})	0.0962	0.1	[0.0905, 0.1019]
Infrequent paths (Π^{In})	0.4983	0.5	[0.4859, 0.5107]
Tree generation time (seconds)	0.89	/	[0.88, 0.90]
Data generation time (seconds)	0.006	/	[0.0059, 0.0065]
Log generation time (seconds)	5.4098	/	[5.3002, 5.5195]

Table 3.15: Descriptive Statistics of a Sample from Population MP_{Data}. Legend for the colors of the cells that describe the confidence intervals (CI): green if CI contains the population value, red if CI does not contain population value, and no color if there is no corresponding population value.

sharpened beyond the workaround for the simulator proposed in Section 3.4.2.2.

Furthermore, the tree, data, and log generation times are recorded in Table 3.15. These performances show relatively fast generation times. The data extension influences the log generation time compared to the log generation time in Table 3.14. This is caused by the checking of case attribute values at each routing decision in the process tree as described by Algorithm 10.

The *randomness* requirement implies that the generation of the trees, data, and logs should be done in a random way. Both the ProM and Python tool implementations support such a random generation. The subsection describing the tool implementation already mentioned that both tools meet the *formats* requirement. Therefore, we can conclude that the “PTandLogGenerator” artefact fulfills all the predefined requirements and “DataExtend” artefacts fulfills all the predefined requirements except for the soundness requirement in a limited amount of situations which is mentioned as threat to the validity in Section 3.6. Table 3.16 provides the reader with an overview of the fulfilled requirements.

R1 Full Control	"PTandLogGenerator" + "DataExtend"
Number of activities	✓
Sequence (WCP-1)	✓
Parallel (WCP-2-3)	✓
Choice (WCP-4-5)	✓
Loop (WCP-21)	✓
Or (WCP-6-7)	✓
Silent (skipping) activities	✓
Reoccurring (duplicate) act.	✓
Long-term (LT) dependencies	✓
Infrequent paths	✓
Routing decision determinism	✓
Soundness	✓[1]
No. traces	✓
Noise	✓
R2 Randomness	
Random Generation	✓
R3 Standard Formats	
Models	✓
Logs	✓
ProM integration	✓

[1] "DataExtend" fulfills the soundness requirement except for a limited amount of situations which is handled by using a workaround in the simulator described in Section 3.4.2.2.

Table 3.16: Evaluating "PTandLogGenerator" + "DataExtend" on the requirements of GED

3.5.3.2 Scalability of PTandLogGenerator

This subsection describes an analysis done in order to assess the scalability of the control-flow tree generation. We particularly choose to focus on the model generation with LT dependencies as this is the major difference with existing data generators. This part of the evaluation studies the relation between tree generation time and control-flow model population parameters. The unfolding of a tree into the unfolded choice tree is the most expensive operation in terms of computation time. Such unfoldings happen when choice and loop constructs appear in the tree and the probability to insert LT dependencies is larger than 0 (see Section 3.2.4). Therefore, 1000 model populations are specified with varying probabilities and settings ($\text{MP}_{\text{scalability}}$ in column 4 in Table 3.13):

- The probabilities of “sequence”, “choice”, “parallel”, “or” and “loop” vary between 0 and 1 while ensuring the sum is equal to 1.
- The probability of LT dependencies varies between 0 and 1.
- The unfolding of loops with choices in the first or second child has a probability equal to 50%
- If loops are unfolded, then the maximum number of repititions of the loop varies between 0 and 2

From each of the 1000 model populations, one random tree is generated. The tree generation aborts after 10,000 seconds. In total 23 trees, i.e. 2.3% of all generated trees, were aborted. The other 977 trees have a median generation time of 0.63 seconds and a minimum and maximum of respectively 0.03 and 8736 seconds. We want to understand which model parameters influence the long tree generation time, and which model parameters lead to exceeding 10,000 seconds for tree generation. Therefore, the spearman correlation coefficients were calculated. Table 3.17 shows that there are only 4 small, yet significant positive correlation coefficients using a 5% significance level. When the probability of a loop construct or the maximum number of loop repetitions increases, then the probability of exceeding 10,000 seconds for tree generation tends to increase. Similarly, when the probability of a choice construct or the maximum number of

Variable 1	**Variable 2**	**Spearman correlation**	**P-value**
Loop ($\Pi^{\circlearrowleft}$)	Aborted	0.12	1.19e-04
Max repeat	Aborted	0.19	1.43e-09
Choice ($\Pi^{\times}$)	Time	0.32	1.39e-24
Max repeat	Time	0.19	7.73e-40

Table 3.17: Positive significant correlations between tree generation time ('Time') or aborting tree generation ('Aborted') and model population parameters.

loop repetitions increases, the tree generation time tends to increase. Overall, it is hard to predict a long tree generation time using only model parameters. One could assign more computing time or use statistical techniques that can handle missing values, e.g. truncated data analysis, to handle the exceptionally long tree generation times.

In comparison, trees without LT dependencies never suffer from exceptionally long generation times. An additional experiment specified another 1000 model populations without LT dependencies and varying probabilities of "sequence", "choice", "parallel", "or", and "loop" as before. Again one tree is generated from every model population. Each of those trees could be generated within 2 seconds. All performances were accomplished on a laptop with an Intel Core i5-4200U processor and 8 GB of RAM memory.

Although "DataExtend" can also introduce LT dependencies to process trees by assigning case attributes to routing decisions that refer to previous routing decisions (see Section 3.4.2), we did not include a scalability analysis of "DataExtend" as an alternative for generating process trees with LT dependencies using the "PTandLogGenerator". That is because "DataExtend" restricts the inclusion of LT dependencies to choices (routing decisions) that are in strict sequence using the Precedence function (see Definition 3.7). In contrast, the LT dependencies introduced by the "PTandLogGenerator" are more flexible. To illustrate this, consider process tree PT_3 in Figure 3.20. "DataExtend" cannot introduce a dependency between the two choices in the tree as they are in parallel and not in sequence. "PTandLogGenerator" on the other hand, can transform it to

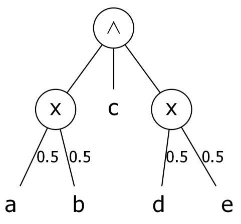


Figure 3.20: PT_3

the unfolded choice tree $s(PT_3^{\times}) = \times\langle\wedge\langle a,c,d\rangle,\wedge\langle a,c,e\rangle,\wedge\langle b,c,d\rangle,\wedge\langle b,c,e\rangle\rangle$ and remove branches to insert random LT dependencies between the two original choices.

3.5.3.3 Effectiveness of the PTandLogGenerator

The final part of the evaluation asserts the effectiveness of the "PTandLogGenerator". It tests whether the additional model constructs allowed by the "PTandLogGenerator", i.e. "or", reoccurring activities and LT dependencies, lead to new insights that could not be obtained by using all model constructs supported by the current state-of-the art technique PLG2 [17]. This test can be formulated in the following hypothesis: discovery algorithms will perform differently in terms of discovered model quality on event logs containing the additional constructs compared to event logs containing all constructs supported by current state-of-the-art. For this purpose, an empirical evaluation with four discovery algorithms, α_{++} [129], ILP [108], Inductive [67] and Flexible Heuristics [127], on two model populations has been done. The first model population ($\text{MP}_{\text{existing}}$) contains models with all constructs supported by PLG2, the second model population (MP_{new}) additionally contains the constructs "or", reoccurring activities and LT dependencies as supported by "PTandLogGenerator". Columns two and three of Table 3.13 display the specific parameter settings for each of the constructs. Notice that the proportions between the constructs sequence, choice, parallel and loop constructs is kept constant, e.g. $\Pi^{\times}/\Pi^{\rightarrow} = 0.5$. Furthermore, notice that we do not compare PLG2 directly with "PTandLogGenerator" as they define the model populations differently, e.g. the size of the models in PLG2 is specified in terms of maximum nested process constructs. As such, a direct comparison may lead to differences that are not due to the additional constructs, but rather the consequence of other process characteristics such as size that we cannot control for.

The evaluation first draws a random sample of 50 models from each population. Then, one log per model is simulated containing 1000 traces and 10% of noise using a combination of the noise operators in Section 3.3.2. Then, all four discovery algorithms mine a model from each log and the quality of each discovered model with regard to that log is measured in terms of fitness and precision using the alignment based fitness and precision metrics [102].

The fitness and precision scores for each discovered model are visualized in a scatterplot: Figure 3.21(a) visualizes the results for the sample from $\mathrm{MP}_{\mathrm{existing}}$, and Figure 3.21(b) visualizes the results for the sample from $\mathrm{MP}_{\mathrm{new}}$. The points situated in the top-right corner represent "good" models, i.e. models that score high on fitness and precision, while points in the bottom-left corner represent "bad" models, i.e. that score low on both fitness and precision. The scatterplots reveal some differences between the miners for the two populations: the heuristics miner has more "good" models in $\mathrm{MP}_{\mathrm{existing}}$ than $\mathrm{MP}_{\mathrm{new}}$, while the opposite seems to be true for the α_{++} miner. To verify these differences statistically, we conduct statistical tests on the differences in fitness, precision, and the combined F_1 score, i.e. the harmonic mean of fitness and precision: $\frac{2 \cdot precision \cdot fitness}{precision + fitness}$ similar to [32].

Table 3.18 shows an overview of the obtained results: column two contains the results for $\mathrm{MP}_{\mathrm{existing}}$, while column three shows the results for $\mathrm{MP}_{\mathrm{new}}$. For each quality dimension the average rank for each discovery algorithm is shown. The algorithms are sorted with the best performing algorithm (with the highest rank) on top. The Friedman test [34] is applied to determine whether there is a significant difference in performance of the discovery technique. The results indicate that the techniques do *not* perform equivalently for each combination of quality dimension and dataset, i.e. the null hypothesis is rejected using a 95% confidence interval. This is followed by a Wilcoxon signed rank test [11, 34] to test the significance of each pairwise difference between algorithms using a Bonferroni corrected significance level to guarantee that the family-wise Type I

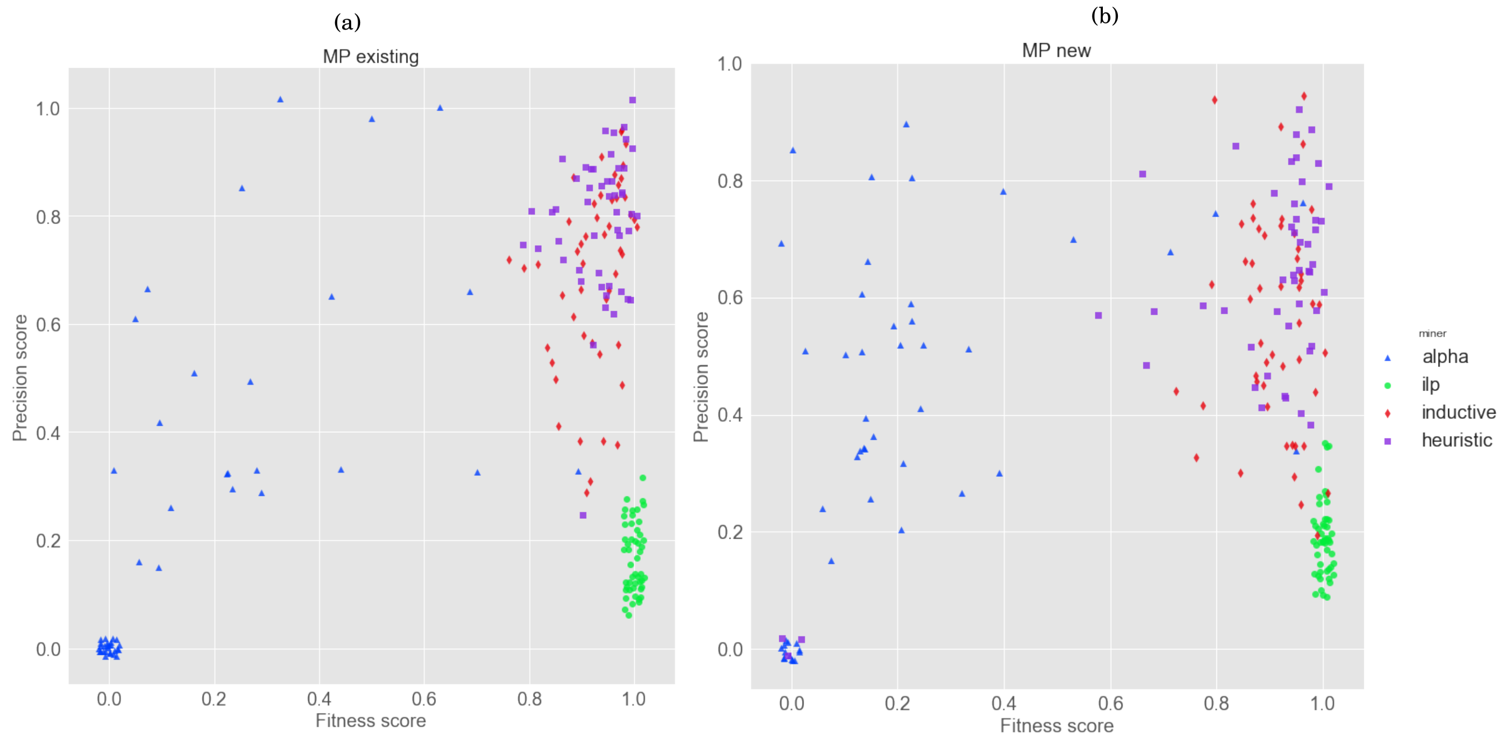


Figure 3.21: Scatterplots of the fitness and precision scores of the samples from: (a) $MP_{existing}$, (b) MP_{new}. Minimum jitter has been added to reduce overlap of observations.

Quality metric	**MP$_{\textbf{existing}}$**	**MP$_{\textbf{new}}$**
Fitness	ILP (4.0)	ILP (4.0)
	Heuristics (2.54)	Inductive (2.52)
	Inductive (2.46)	Heuristics (2.42)
	α_{++} (1.0)	α_{++} (1.06)
Precision	Heuristics (3.6)	Heuristics (3.36)
	Inductive (3.18)	Inductive (2.94)
	ILP (1.62)	α_{++} (2.28)
	α_{++} (1.6)	ILP (1.42)
F_1	Heuristics (3.68)	Heuristics (3.5)
	Inductive (3.28)	Inductive (3.2)
	ILP (1.78)	ILP (1.9)
	α_{++} (1.26)	α_{++} (1.4)

Table 3.18: Average rankings for process discovery algorithms for each quality dimension within a model population. Pairs of techniques that do not differ statistically from each other at the 95% confidence level are underlined.

error is smaller than 5%. Pairs of techniques that do not differ statistically are underlined.

In the fitness dimension the order between Heuristics and Inductive miner is different for the two datasets. However, the difference between these two miners is not statistically significant. In the precision dimension the order between ILP and α_{++} miner is different for the two datasets, yet the difference between the algorithms is not statistically significant for the MP$_{\text{existing}}$ dataset. Also in the precision dimension, the difference between Heuristics and Inductive miner is only statistically significant for the MP$_{\text{existing}}$ dataset. Looking beyond the average rankings, the Heuristics miner outperforms Inductive miner 36 times for the MP$_{\text{existing}}$ dataset while it decreases to 33 times for the MP$_{\text{new}}$ dataset. Finally, all differences in terms of F_1 between miners are statistically significant for the MP$_{\text{existing}}$ dataset, while for the MP$_{\text{new}}$ dataset the difference between Heuristics and Inductive miner is not statistically significant.

Overall, the conclusion of the analysis is that the difference between Heuristics and Inductive miner becomes smaller in terms of precision for models with “or”, reoccurring activities and LT dependencies. Conversely, the difference between α_{++} and ILP in terms of precision becomes larger for such models. These observations show that the extra constructs have negative effects on the Heuristics, Inductive and ILP miner (only on precision), while it has positive effects on the α_{++} miner. Moreover, the negative effects on Heuristics miner are larger than the negative effects on Inductive miner. As such, these observations provide evidence for the hypothesis that discovery algorithms perform differently on event logs with the additional constructs compared to event logs using all constructs supported by current state-of-the art technique PLG2 [17]. This demonstrates the effectiveness of the “PTandLogGenerator”.

3.6 Limitations and threats to validity

Although the above evaluation highlighted that the requirements of the “PTandLogGenerator” and “DataExtend” generators are met and illustrated the scalability and effectiveness of the “PTandLogGenerator”, several limitations and possible threats to the validity remain.

The first limitation involves the use of process trees as modeling notation in the “PTandLogGenerator” and “DataExtend” generators. The use of process trees offers two important advantages: the soundness property avoids deadlocks during simulation, and their block-structuredness allows to build trees in a stepwise manner while controlling for the workflow patterns. On the other hand, however, one could argue that the BPMN modeling language [46] is “richer” as it allows to express more complex workflow patterns and integrates other process perspectives such as resources and decisions (using the connection with DMN [47]). As such, the use of BPMN would enable a wider range of possible extensions than process trees, but also presents the challenge to deal with possible deadlocks during simulation.

A second limitation and possible threat to the validity regards the interplay between control-flow and data process perspectives in “DataExtend”. As indicated earlier, “DataExtend” considers only exclusive choices (XOR), and not loops and “or” (multi-choice) as routing decisions. Furthermore, the interaction between data and control-flow at the routing decisions is limited to the constant values of the case attributes and the chosen branches in previous routing decisions. This excludes for example the last executed activity as a case attribute or time-related attributes that change during case execution which could influence a routing decision.

Lastly, the insufficient soundness constraints of “DataExtend” are a possible threat to the validity. The soundness constraints to prevent deadlocks and dead activities in Section 3.4.2 are necessary, yet not sufficient. This means that the simulator has been adapted to circumvent a possible deadlock by assuming that all outgoing branches are possible when the state of a case does not match with any of the routing decision rules. Furthermore, dead activities can occur, although in a limited number of situations: if there are two routing decisions $\times_1$ and $\times_2$ such that $\times_2$ depends on $\times_1$ and assigned case attributes overlap, i.e. $\text{Assign}(\times_1) \subseteq \text{Assign}(\times_2)$.

3.7 Conclusion

The comparison of process discovery and decision mining algorithms has gained importance in the process mining research domain. Typically, such a comparison requires an empirical analysis that involves large experiments. Yet, little research has been performed on how to collect the appropriate event data (models and event logs) as input for the empirical analysis. A clear methodology for acquiring such data and an implementation thereof are lacking.

This chapter firstly introduces the GED methodology and “PTandLogGenerator” to generate artificial event data for empirical process discovery analysis. It involves three steps: define a model population, draw a sample of models from that population and simulate the sample of models into a sample of event logs. The demonstration and evaluation show that the “PTandLogGenerator” succeeds in generating artificial data for empirical process discovery analysis such that:

- the generated models are random samples of predefined populations, allowing for a wide range of suitable (confirmatory) statistical experimental analysis,

- the populations allow for more complex process models than possible by the existing approaches (including LT dependencies, “or” and reoccurring activities),
- the approach is performant enough for large scale experiments,
- the approach is able to reveal insights which remained hidden when considering simpler process populations (which were only possible so far by existing techniques).

The “PTandLogGenerator” in this chapter does not claim to be complete with regard to full control over all possible control-flow patterns. However, it includes all patterns that were frequently used in process discovery comparisons. Moreover, the definition of LT dependencies in this paper focuses on dependencies between exclusive choices, in future work this definition could be extended to allow for dependencies between non-exclusive choices.

Secondly, this chapter introduces an extension to the “PTandLogGenerator” called “DataExtend” to include case attributes that explain the routing decision logic. It first extends the routing decisions that correspond to exclusive choices in a control-flow model with random decision rules based on case attributes. In a next step, the extension simulates the model and rules into an event log with case attributes. The demonstration and evaluation shows that “DataExtend” allows the generation of random data-extended models and event logs where a user can control for the determinism of routing decisions. The generated models and event logs can be used to empirically evaluate decision mining algorithms. In future work, the set of routing decisions considered by “DataExtend” could be increased to also include loops and “or” (multi-choice) workflow patterns. Also, the soundness constraints should be sharpened to guarantee decision-aware soundness in all cases.

CHAPTER 4

AN INTEGRATED EVALUATION PROCEDURE FOR PROCESS DISCOVERY ALGORITHMS

The abundance of discovery algorithms has made it increasingly important to develop evaluation procedures that can compare the quality of these discovery techniques in rediscovering the underlying process, especially in terms of balancing between overfitting and underfitting. As detailed in Section 2.2.1, several comparison approaches have already been proposed in literature. Unfortunately, these approaches are characterized by at least one of the following four major limitations related to the unsolved challenges in Section 2.2.2:

1. The quality measurement is not independent from the modeling notation in which the discovered models are represented, e.g. two behaviorally equivalent models may have very different precision scores, or quality can only be measured after a conversion that does not preserve the behavior precisely. This restricts the evaluation to a comparison of the algorithms that generate models in one specific notation (see [6, 33, 86, 87, 89, 115, 125]).

2. The evaluation results are based on real event logs and cannot be generalized as the population of processes from which they originate is unknown (see [6, 33, 87]). Processes come from different populations depending on the type of behavior allowed. Processes may have different behavioral characteristics, with parts that can repeat, with mutually-exclusive and parallel branches, with LT dependencies and so on. Also, these characteristics can be more or less predominant in a process model. Different algorithms may deal better with certain characteristics than others. And the quality of the discovered model may also depend on the predominance of certain characteristics. Performing a comparison without acknowledging the influence of these behavioral characteristics can lead to incorrect conclusions.

3. Existing comparison approaches use manually created processes to generate artificial event data (see [33, 86, 87, 89, 115, 125, 126]). As a result the studied process characteristics are not randomly included in the processes. Furthermore, relatively few processes and event logs were created. This prevents the results from being statistically and generally valid.

4. Existing comparison approaches apply log-based quality measurement to estimate the discovery algorithm's quality to rediscover the underlying process (see [6, 33, 86, 87, 89, 115, 117, 125]). However, these log measures only provide unbiased results if the log is complete and contains no noisy behavior. This is partly caused by the fact that the same data is used both to discover the model and measure its quality, a problem that is also encountered in machine learning [44].

To overcome these limitations, this chapter describes an integrated process discovery evaluation procedure (as stated in the second research goal in Section 1.2) that:

1. Abstracts from the modeling notation employed during quality measurement;

2. Starts from the definition of a process population where the distribution of several behavioral characteristics is known. From this population a random sample of process models and event logs is drawn, thus making it possible to evaluate and generalize the influence of behavioral characteristics on the quality of the discovered models by the different algorithms under analysis;

3. Performs experiments on random samples of a user-specified size, so as to return statistically valid results;

4. Uses the knowledge of the reference models to get unbiased estimates of a discovery algorithm's quality to rediscover the underlying process.

Repeating the evaluation for large amounts of process models and event logs would be time-consuming and error-prone when done manually. Therefore, the steps of the evaluation procedure, i.e. data selection, process discovery, quality measurement, and statistical tests are integrated in a workflow to allow the automation of evaluation experiments. The automation together with the above contributions correspond to the requirements for the process discovery evaluation procedure as discussed in Section 2.3.2.

The material in this chapter is based on the journal paper "An Integrated Framework for Process Discovery Algorithm Evaluation" [56] and lessons learned from the Process Discovery Contest [22]. The chapter starts with a discussion on the design of the new process discovery evaluation procedure. Then, the chapter will discuss the implementation of the procedure, followed by a demonstration and evaluation of the procedure. Finally, a conclusion summarizes the chapter.

4.1 Integrated discovery evaluation procedure

The proposed procedure aims to evaluate the quality of discovery algorithms to rediscover a model when confronted with a fraction of its behavior. The procedure is designed based on the principles of scientific workflows and experimental design to fulfill the last two groups of requirements in Table 2.5 . The former captures the complete evaluation experiment in a workflow that can be automated, reused, refined and shared with other researchers [7]. The latter allows for precise answers that a researcher seeks to answer with the evaluation experiment [64]. The fundamental principles of experimental design, randomization, replication, and blocking, were already discussed in Section 2.3.2.2.

To integrate the steps needed for empirically evaluating process discovery algorithms, the procedure is built as a scientific workflow. Generally such workflows are represented as a directed graph with nodes denoting computational steps and arcs expressing data flows and data dependencies between steps [75] (see Fig. 4.1). Bolt et al. [14] have described generic process mining building blocks to conduct process mining experiments using scientific workflows.

Scientific workflows offer several advantages over traditional ways to conduct process discovery evaluation. The first advantage comes from workflow automation. Experiments evaluating discovery techniques involve large-scale and computationally expensive experiments that require intensive human assistance [14]. Therefore, automating these experiments removes the need for human assistance and reduces the time needed to perform experiments. A second benefit comes from the modularity of the workflows. This allows researchers to adapt and extend an existing workflow, e.g. by using other parameter settings or adding new process discovery techniques. A final benefit of scientific workflows is that they can be shared with other researchers. As a result other researchers can replicate experiments with little effort. In this way, our procedure facilitates repeated process discovery evaluation, e.g. it becomes trivial to evaluate another set of algorithms or to assess the algorithm's performance with regard to other data characteristics (e.g. noise, control-flow patterns, etc.).

The remainder of the section will discuss the design of the procedure as a workflow and its building blocks that implement the principles of experimental design in more detail.

4.1.1 Design and use of the evaluation procedure

The procedure focuses on evaluating control-flow discovery algorithms. Therefore, other process related perspectives, such as data and resources, are out

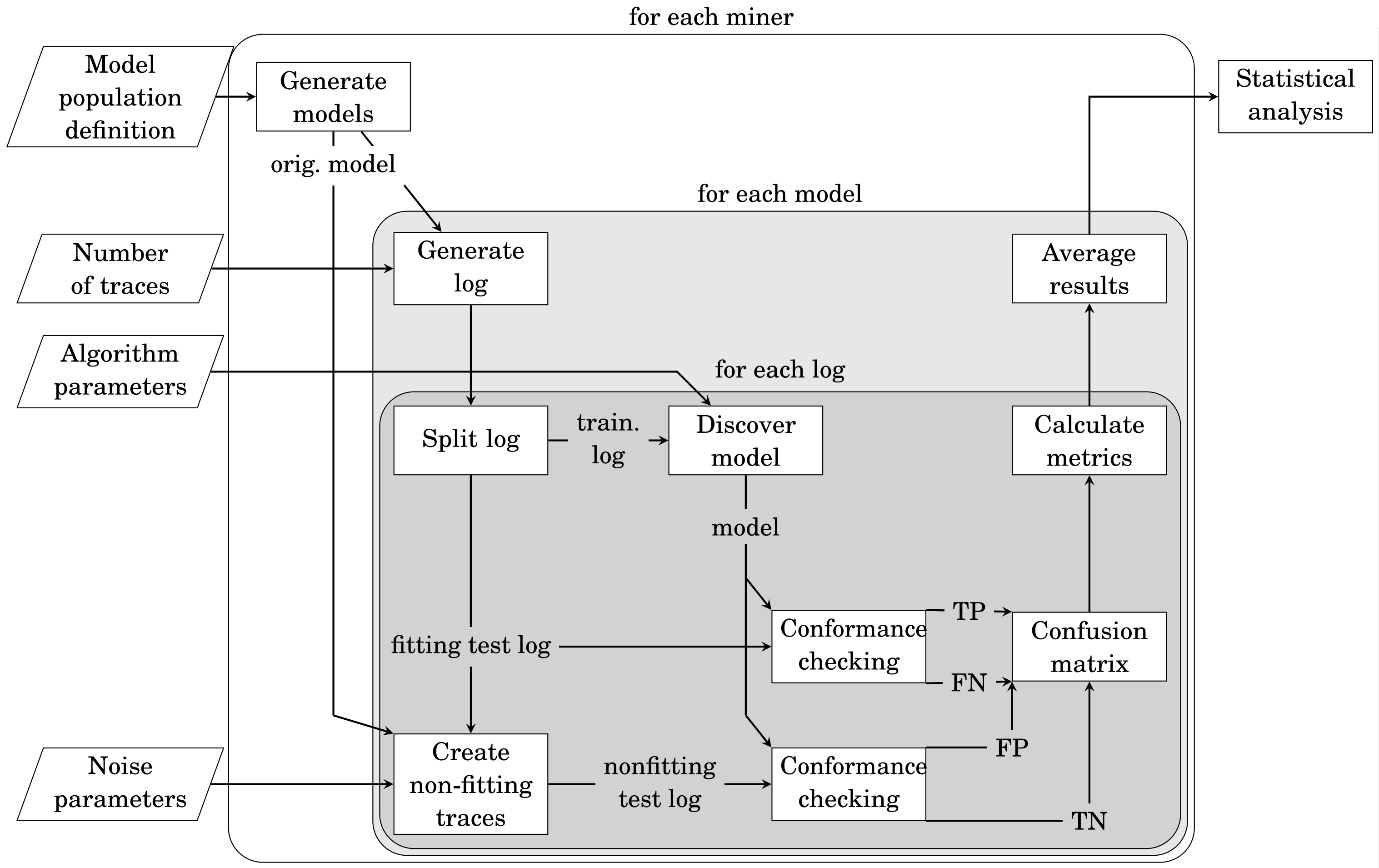


Figure 4.1: Overview of the integrated process discovery evaluation procedure

of its scope.[1] Moreover, the procedure aims at evaluation instead of predicting the best performing algorithm given an event log. The procedure enables two main objectives: benchmarking different discovery algorithms, and performing sensitivity analysis, i.e. what effect does a control-flow characteristic or event log characteristic have on algorithm performance.

Fig. 4.1 illustrates the design of the new evaluation procedure as a workflow. The directed graph shows how the different tasks needed for evaluating process discovery algorithms are connected. The procedure starts from a predefined evaluation objective and enforces the consecutive execution of data generation, process discovery, quality measurement and statistical analysis. The procedure applies a classification approach to allow for the evaluation of discovery algorithms generating models in different notations. It is assumed that the notations have clear and formal semantics such that a trace can be replayed on the model to determine whether the model allows the trace or not.

The first step, i.e. the data generation, is triggered by the objective of the experiment. As a result, the objective determines the control-flow behavior a researcher wants to include in the event logs. The specification of control-flow behavior defines a model population. This population definition is the start of the data generation phase (cf. Section 3.2). For example, the objective is benchmarking several discovery algorithms on event logs containing basic process behavior and behavior that is typically hard to rediscover for current state-of-the-art algorithms: silent activities, reoccurring activities, infrequent paths, and long-term dependencies. As such we define a model population with probabilities larger than zero for each of the basic workflow and "hard" to rediscover patterns larger than zero, e.g. the model population $MP_{example}$ in Table 4.1.

For each discovery algorithm to be tested, multiple instances of the "generate models" task run in parallel. The generation results in multiple *randomly* sampled process models from the same population. Each model ("original model") is then simulated by the task "generate event log" to create one event log, i.e. a *random* sample of traces from all possible traces allowed by the model (i.e. no

[1]However, the same ideas can be applied to include these other perspectives as will be shown in Chapter 5.

Parameter	**Population $\mathbf{MP_{example}}$**
Number of visible activities	(10,20,30)
Sequence ($\Pi^{\rightarrow}$)	0.5
Parallel ($\Pi^{\wedge}$)	0.15
Choice ($\Pi^{\times}$)	0.25
Loop ($\Pi^{\circlearrowleft}$)	0.05
Or ($\Pi^{\vee}$)	0.05
Silent activities (Π^{τ})	0.1
Reoccurring activities (Π^{Re})	0.1
Long-term dependencies (Π^{Lt})	0.5
Unfold loops	True
Max repeat (k)	1
Infrequent paths (Π^{In})	0.5
Determinism level	n/a

Table 4.1: Example model population parameters.

noise is added). The samples of process models and event logs constitute as "the ground truth". The "original model" or reference model represents the underlying process which the discovery algorithms will try to rediscover using the event log. By testing algorithms on multiple randomly drawn event logs (each log is drawn from a randomly drawn model), the design implements the randomization and replication principles of experimental design (see Section 2.3.2.2). This enables us to accurately assess the effect of an algorithm on model quality and generalize these findings to the model population.

Next, the procedure applies k-fold cross-validation to measure the quality of the discovery algorithm to rediscover the underlying process. By using k-fold cross validation the obtained estimate is less likely to suffer from bias, i.e. it helps to decrease the difference of the estimate from the real unknown value of the algorithm's quality on the model population. As commonly used in machine learning we propose to set k equal to 10 which is a compromise between obtaining less-biased estimates and acceptable computation time [51]. The 10-fold cross-validation splits each event log into ten subsets, i.e. folds, of equal sizes. Nine folds form the training log, while the remaining fold serves as the test log. The task “discover model” applies the algorithm to induce a model from the training log.

To this point, the test log only contains positive examples, i.e. traces that fit the original model (no noise is added). The classification approach requires also negative examples, i.e. traces that do not fit the original model. To generate negative examples, the task “create non-fitting traces” alters half of the test traces until they cannot be replayed[2] anymore by the original model (“the ground truth”) to create non-fitting traces. Thus, the test log contains a 50/50 balance between fitting and non-fitting traces to avoid the class imbalance problem which makes the evaluation more difficult [52].

Subsequently, the procedure measures the quality of the discovery algorithm by using the discovered model to classify the test traces. This classification happens within the “conformance checking” block which replays all traces on the discovered model. A trace representing real process behavior should be classified as allowed, i.e. completely replayable. A trace representing behavior not related to the real process should be classified as disallowed by the discovered model, i.e. not completely replayable. This approach allows for any discovery algorithm generating models with clear and formal replay semantics.

The classification results are then combined in a confusion matrix (see Section 4.1.2). Based on that matrix, one can compute the well-known recall and precision metrics to evaluate the quality of the discovery algorithm. These quality measures are used because the procedure focuses on the traces that are completely replayable by the discovered model. More specifically, recall measures how much fitting behavior in the test log is classified as fitting by the discovered model, and precision measures how much behavior classified as fitting by the discovered model is actual fitting behavior in the test log. Good recall is important as every discovery algorithm tries to discover a model containing the behavior of the underlying process [33]. On the other hand, precision balances between underfitting and overfitting [99]. The procedure repeats the process of splitting, discovery, creating non-fitting traces and conformance checking ten times, each time with a different fold as “test log”. The task “average results”

[2]Replay uses the trace and the model as input. The trace is “replayed” on top of the model to see if there are discrepancies between the trace and the model [99].

computes the average of the metric values over the ten folds to get an estimate of the algorithm's quality to rediscover the underlying process. Finally, the task "statistical analysis" tests the hypotheses formulated in the context of the objectives.

This procedure's design has the property that no two discovery algorithms are applied on the same event log. Furthermore, for each generated model - randomly drawn from a predefined model population - we randomly draw only a single event log. Consequently, all discovered models and any corresponding quality metric are independent observations which is an important assumption underlying many standard statistical techniques. We acknowledge that this design decision is not the only option as one could test discovery algorithms on the same logs. This alternative design would have more statistical power for the same sample size as it removes the variance between logs from the error terms[3] used to test the effects of discovery algorithms [45]. However it requires more complex statistical techniques to deal with the dependency between observations. We can compensate for the loss in power in our design by increasing the sample size. One can define the required sample size based on the desired power of the statistical analysis in advance. The power of a binary hypothesis test is the probability that the test correctly rejects the null hypothesis when in fact the alternative hypothesis is true.

Finally, the design of the procedure is based on the experimental design principles which enables users to obtain algorithm's quality measures that are *independent* from specific process models and event logs. More specifically, this starts from the generation of (preferably large) random samples of process models and logs from a population, which are then used to estimate the quality of a algorithm with regard to that population. This contrasts evaluation based on a small non-random sample of (manually created) process models and event logs as it could influence the quality estimate to only reflect these particular models and logs.

The following subsection will elaborate on each of the tasks in the procedure.

[3]The error term or residual is a variable in the statistical model that is created when the model does not fully capture the actual relationship between independent and dependent variables.

4.1.2 The building blocks of the procedure

The building blocks are discussed in order of appearance in the procedure.

4.1.2.1 Generate models

This building block generates a random sample of process models from a model population as specified by the GED methodology (see Section 3.1). The input of this block is a model population which a user defines by assigning probabilities to each of the model building blocks, i.e. control-flow patterns, and setting the size of the models in terms of visible activities[4]. The probabilities of the control-flow patterns influence the probability for each pattern to be included in the resulting process model. For example, if the probability of loops is 0.2, then on average 20% of the model constructs will be of type loop.

In particular, this block allows one to generate process models that can feature the basic workflow control-flow patterns identified in [92], namely:

- *Sequence*: certain process activities need to be sequentially executed.
- *Exclusive choice*: certain process parts/branches of the process are mutually exclusive. In several notations, this is known as XOR split/join.
- *Parallelism*: certain parts/branches are "parallel", indicating that all branches will be executed but in no particular order. In several notations, this is known as AND split/join.
- *Or*: certain parts/branches are in multi-choice ("or"), indicating that a subset of these branches will be executed. Differently from exclusive choice, multiple parts can be executed in parallel; different from the parallelism construct, not every part that follows the reached point needs to be executed. In several notations, this is known as OR split/join.
- *Loop*: certain parts of the process can be sequentially repeated multiple times.

[4]Notice that a population MP as specified in Definition 3.3 does not include the log characteristics such as number of traces and noise.

This set of pattern is complemented by a number of more advanced patterns:

- *Silent activities*: certain activities are inserted into the model for a process-routing purpose. For instance, combined with exclusive choices, silent activities enable certain parts of the process to be skipped.
- *Reoccurring activities*: the same activity appears in different parts of the process, indicating that the activity can reoccur.
- *Long-term dependency*: the choice of one or multiple branches at a certain moment in the execution of the process can influence which choices become available at a later point.
- *Infrequent Paths*: this is always combined with an exclusive choice. When the execution reaches an exclusive choice, certain potential process branches have a higher probability to be chosen. In fact, this pattern is rather related to the generation of event logs.

These constructs are those typically discovered by discovery algorithms because they are the most relevant. BPMN [46] and other modelling notations support more complex constructs, such as multiple instances and terminating events; however, at the best of our knowledge, no discovery algorithms support their discovery.

As a result, this block allows users to fully control the control-flow behavior in the generated models and generalize the results to the predefined population. The user defines a population MP as specified in Definition 3.3: $MP =$ (minimumVisibleAct, modeVisibleAct, maxVisibleAct, Π^{Base},Π^{τ},Π^{Re},Π^{Lt},Π^{In}).

4.1.2.2 Generate log

For each generated model the “generate log” block creates an event log, i.e. a random sample of all possible traces allowed by that model. This building block uses the given process model to generate a user-specified number of traces per event log. The exclusive choices in each of the process models have output-branch probabilities. As a result, the resulting event log contains a random set of fitting and complete traces. The presence of infrequent paths will make some traces more probable than others which will result in event logs with infrequent behavior.

4.1.2.3 Split log

This building block applies the first step needed for the 10-fold cross validation evaluation method. The step splits a given event log into ten subsets (folds) of equal size. Nine folds form the "training log" and are the input of the discovery algorithm. The tenth fold is the "test log" which is split in half: one half constitutes the "fitting test traces", the other half will serve as input of the "create non-fitting traces" block to make "non-fitting test traces". This is repeated ten times such that each of the ten folds becomes a "test log" exactly once.

4.1.2.4 Create non-fitting traces

In a classification approach the "test log" should contain positive and negative examples. To this point, there are only positive examples, i.e. traces that fit the original model. The "Create non-fitting traces" building block alters the given test traces so that they do not fit the original model anymore. The goal of the non-fitting traces is to punish overgeneralization of discovery algorithms. The flower model is an example of extreme overgeneralization that allows every possible trace involving the set of activities but provides no added value in a business context [99]. Therefore, the new procedure aims to punish typical overgeneralizing patterns: unnecessary loops, activity skips and parallelism, by altering the traces using specific *noise operations* (see description below) that can add or remove behavior. The traces are altered but kept as close to the original trace as possible. In this way, the procedure avoids non-fitting traces that would be trivially rejected by overgeneralizing models.

Given a process model and a set of fitting traces, noise is added to each trace as follows. First, one or more of the following noise types based on [41], is added with a user specified probability:

- *Add activity*: one of the process activities is added in a random position within the trace. A special case of this is when an activity is duplicated, i.e. inserted immediately after the original. This type of noise aims to punish unnecessary loops, i.e. a certain activity that can be repeated in the discovered model, but cannot be repeated in the original model.

- *Remove activities*: one or more activities are randomly removed from the trace. This type of noise targets the detection of activity skips in the discovered model that are not allowed in the original model.

- *Swap random activities*: swap the two randomly chosen activities in a pair. A special case of this is when two consecutive activities are swapped. This type of noise aims to punish unnecessary parallellism between certain activities in the discovered model while in the original model those activities should be executed in a particular order.

Then, the modified trace is checked for fitness with respect to the original model. If the trace does not fit anymore, it is a noisy trace which will not be edited anymore. If the trace still fits the model, noise is added again (and checked afterwards) until it does not fit anymore, or until noise has been added five times. If the noisy trace still does not fit the model, the trace is discarded and another trace is randomly selected from the set of fitting traces. This trace follows the same process as was described above.

Table 4.2 illustrates the different types of overgeneralization that the evaluation procedure will punish for. The columns of the table contain the original models, the training logs generated from the original models, and the discovered models with an overgeneralization pattern. The first row contains the original model with activities “b” and “c” in an “or” such that each trace will contain one of those activities or both in no particular order. The training log contains three traces: $\langle a,b,d\rangle$, $\langle a,c,d\rangle$, and $\langle a,b,c,d\rangle$. Given the training log, the ILP miner [108] discovers a Petri net with loops on both activities “b” and “c”. Those loops are unnecessary as they allow for multiple executions of “b” and “c”, which are not allowed by the original model. This overgeneralization can be punished by using the *add activity* noise operator: duplicate activity “b” in the fitting trace $\langle a,b,d\rangle$ to get the trace $\langle a,b,b,d\rangle$ which is non-fitting with regard to the original model, but is allowed by the discovered model due to the unnecessary loop on activity “b”. This creates a false positive and lowers the precision score of the discovered model (discussed in the following sections).

The second row of Table 4.2 includes the original model with a choice and a sequence in parallel. The training log consists of five out of six possible traces and the Inductive miner [69] discovers the model with skips on activities “a” and “b”. Those activity skips are unnecessary as they allow to skip both activities in a trace whereas in the original model always “a” or “b” needs to be executed. The *remove activities* noise operator allows to punish the unnecessary activity skips: remove activity “a” from $\langle a, c, d \rangle$ to get the trace $\langle c, d \rangle$ which is non-fitting with regard to the original model, but is allowed by the discovered model due to the unnecessary activity skips on activities “a” and “b”.

The last row of Table 4.2 contains the original model with a duplicate activity “a”. The training log includes the two possible traces according to the original model. Applying the ILP miner [108] on the training log results in the Petri net with only transitions and no arcs. All three activities can be executed in parallel whereas the activities are not in parallel in the original model. The *swap random activities* noise operator enables the punishment of the unnecessary parallelism: swap activities “a” and “b” in $\langle a, b, a \rangle$ to get the trace $\langle b, a, a \rangle$ which is non-fitting with regard to the original model, but is allowed by the discovered model due to the unnecessary parallelism on activities “a” and “b”.

To illustrate the noise insertion until the modified trace is non-fitting with regard to the original model, consider the first row of Table 4.2. Suppose the fitting trace $\langle a, b, c, d \rangle$ is selected for noise insertion and all three possible noise operators, *add activities, remove activities, swap random activities*, each have the same probabiliy $\frac{1}{3}$ to be applied. Suppose that first the activities “b” and “c” are swapped to get trace $\langle a, c, b, d \rangle$ which is still fitting the original model. Then, the modified trace gets another type of noise, e.g. delete activity “c” to get $\langle a, b, d \rangle$ which is still fitting. Again, noise is added to the modified trace, e.g. duplicate activity “b” which results in $\langle a, b, b, d \rangle$ which does not fit the original model, and hence is a finished noisy trace. In this example, all three noise operators were applied once, but this is not always the case as a certain operator could be applied multiple times which depends on the probabilities the user assigned to each operator.

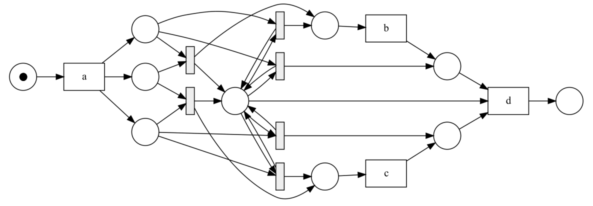
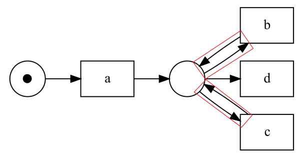
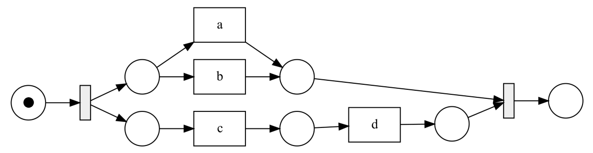
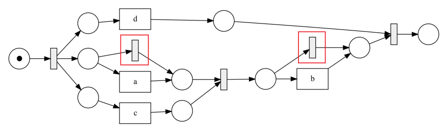
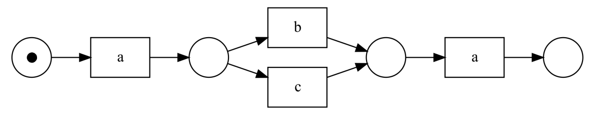
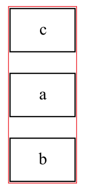
Original model	Log	Overgeneralized discovered model
	$\langle a,b,d \rangle$ $\langle a,c,d \rangle$ $\langle a,b,c,d \rangle$	
	$\langle a,c,d \rangle$ $\langle c,a,d \rangle$ $\langle c,b,d \rangle$ $\langle c,d,a \rangle$ $\langle c,d,b \rangle$	
	$\langle a,b,a \rangle$ $\langle a,c,a \rangle$	

Table 4.2: Illustration of overgeneralizing patterns.

4.1.2.5 Discover process model

This block applies a discovery algorithm to the “training log” to induce a process model. This could be any discovery technique with user specified parameter settings. The discovered model will be used for conformance checking.

4.1.2.6 Conformance checking

The conformance checker will replay the given traces on the discovered model. The trace-level fitness metric will be used as it allows for an unambiguous interpretation of the two possible outcomes: i.e. completely replayable or not-completely replayable. Secondly, the binary outcome naturally allows for the classification approach: if a trace can be completely replayed by the discovered model it belongs to the “fitting” class, otherwise the trace is part of the “non-fitting” class. The number of classes could be extended by adopting an event-level metric to create a more fine-grained evaluation that distinguishes between partially fitting traces. However, we argue that determining the classes for partially fitting traces would require additional research, which is outside the scope of this thesis.

4.1.2.7 Calculating quality metrics

The building blocks “Confusion matrix” and “Calculate metrics” summarize the quality of an algorithm using three standard metrics adopted from the data mining and information retrieval domain [130]: precision, recall and F measure. Traditionally these metrics are based on:

- True Positives: the number of real traces that **fit** the discovered model.
- False Positives: the number of false traces that **fit** the discovered model.
- False Negatives: the number of real traces that **do not fit** the discovered model.
- True Negatives: the number of false traces that **do not fit** the discovered model.

The *precision* metric refers to the percentage of real traces from all the traces that fit the **discovered** model.

$$\text{Precision} = \frac{\text{True Positives}}{(\text{True Positives} + \text{False Positives})}$$

Inversely, the *recall* metric refers to the percentage of traces that fit the **discovered** model from all the real traces.

$$\text{Recall} = \frac{\text{True Positives}}{(\text{True Positives} + \text{False Negatives})}$$

The procedure uses the F_1 variation of the F measure to combine the precision and recall dimensions into a single metric, as suggested by De Weerdt et al. [32], which makes the comparisons between algorithms in both dimensions easier. This statistic refers to the harmonic average of the precision and recall metrics.

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{(\text{Precision} + \text{Recall})}$$

4.1.2.8 Result analysis

The evaluation procedure allows users to compare the performance of algorithms and to study the effect of control-flow characteristics on algorithm performance. The statistical analysis based on the evaluation results depends on the objectives of the experiment and the corresponding hypotheses to be tested. Therefore, the procedure does not incorporate specific statistical techniques, instead it can be used with a whole range of exploratory, descriptive and causal statistical techniques to test any hypothesis that can be expressed in terms of precision, recall, F_1 score, and characteristics of log and model. This will benefit the adoption of the procedure for all types of evaluation studies, rather than serve a specific purpose.

The next section will describe an implementation of the new procedure using a scientific workflow system that allows to automate and share process discovery evaluation experiments.

4.2 Implementation

First, the section will discuss an example implementation for the new evaluation procedure. The second part of the section will discuss how one can add new discovery algorithms that use different modelling notations than supported by the current implementation.

4.2.1 RapidProM implementation

The building blocks used in the new evaluation procedure (described in Section 4.1.2) were implemented as *operators* in the *RapidProM* extension of the *RapidMiner* scientific workflow tool [14, 103].[5] The RapidProM extension provides several operators that support process mining tasks which can be connected to exploit the benefits of scientific workflows [14]. The next part of this section briefly discusses the new and existing operators needed to support each building block in the proposed procedure.

The "PTandLogGenerator" is used for the implementation of the "Generate models" and "Generate log" building blocks in the procedure. Section 3.5 has demonstrated that a user can control all the control-flow patterns specified in Section 4.1.2.1. Additionally, the simulation algorithm of the "PTandLogGenerator" (see Section 3.3.2) allows a user to specify the size of the random set of traces that are drawn from a given process tree.

In a next step a new operator, named "Split Event Log", implements the splitting of the generated log into a training log and a test log. Then, the operator "Generate noisy log" makes half of the traces in the test log non-fitting using the following noise operations: remove one activity from a trace, duplicate an activity, and swap consecutive activities. These operations are specific instantiations of the noise operations specified in Section 4.1.2.4 and they address the most challenging tasks of any discovery algorithm: discover activity skips, loops and parallelism [41].

The discovery task is currently supported by eight operators, each applying a different discovery technique, inter alia the Alpha+ Miner [106], Heuristics

[5]The *RapidProM* extension is open source and can be downloaded at `www.rapidprom.org` or in the *RapidMiner Marketplace*.

Miner [128], ILP Miner [114] and the Inductive Miner [67]. Then, the conformance checking uses the alignment-based fitness technique [102] to check whether a trace perfectly fits a model or not. In contrast to the classic token-based fitness technique [91], alignments are able to deal with silent and reoccurring activities [4, 73]. Since the calculation of an optimal alignment can be time-consuming, the optimized implementation presented in [73] was adapted in order to stop aligning a trace to a model as soon as a move on log or a move on model (except when it corresponds to a silent transition) is mandatory in order to align the trace to the model. By adapting this technique into a “binary” conformance checker i.e., a trace can perfectly fit the model (1) or not (0), we aim to reduce the runtime of experiments by several orders of magnitude. This binary conformance checker is implemented as a new RapidProM operator.

Finally, the outputs of the two conformance checking nodes are combined with existing RapidMiner operators to calculate the true positives, false negatives, false positives and true negatives to form the confusion matrix and calculate the F_1, precision, and recall values. The metric values are summarized in a Comma Separated Values (CSV) format. This format can easily be exported to other tools such as Excel and R to perform statistical tests.

All the operators disccussed above are combined in workflow that executes the new procedure and is shared at `http://data.4tu.nl/repository/uuid:039bc213-b09d-4c96-bc1d-15bc13f645e6`. By sharing this workflow we aim to reduce the setup time of other researchers that use or extend the new procedure.

4.2.2 Extensibility of the RapidProM implementation

As claimed in Section 4.1.1, the new evaluation procedure is not bound to Petri nets or any other modelling notation. As a consequence, it is extensible to incorporate new discovery algorithms, independently of the notations in which these algorithms represent the discovered model. Every change that is necessary to evaluate a new discovery algorithm that produces models in a notation N (e.g. BPMN) is related to the implementation, whereas the procedure workflow does not require changes.

In the RapidProM implementation, it is necessary to: (1) plug-in the new algorithm as a new operator that implements the “Discover model” building block (cf. Section 4.1.2.5) and (2) plug-in a new conformance checker for notation N, with the latter not being necessary if notation N is already among those available in RapidProM. Notice that it is not necessary to change the “PTandLogGenerator” implementation of the building block “Generate models”. Any model generator in any notation that can represent the patterns defined in Section 4.1.2.1, such as process trees, can be employed. These models are only used to generate the event logs with fitting and non-fitting traces and are not directly compared with the models that are discovered.

To illustrate this, consider the case when one wants to evaluate algorithms that discover BPMN models while limiting the number of changes to the RapidProM implementation. The implementation of the algorithm needs to be plugged into RapidProM and a conformance checker for BPMN models needs to be implemented in RapidProM. As a matter of fact, this conformance checker is not necessary as one can convert the BPMN model into a trace equivalent Petri net such that each execution of the BPMN model is possible in the Petri net, and vice versa [39, 62]. In a next step, the Petri-net conformance checker can be employed. The trace equivalence between the BPMN and the Petri net models guarantees that every trace that is diagnosed as fitting/unfitting using the equivalent Petri net will also be as such with respect to the original BPMN model.

The next two sections discuss a demonstration and evaluation of the new evaluation procedure and its RapidProM implementation. Design Science research requires these steps to show that the created artefact, i.e. the new evaluation procedure, can overcome the four limitations of process discovery evaluation listed in the introduction of this chapter.

4.3 Demonstration and evaluation

The demonstration and evaluation of the new evaluation procedure involve two rounds of experiments:

1. The first round will validate that the proposed evaluation procedure effec-

tively supports two main evaluation objectives: benchmarking different discovery algorithms, and performing sensitivity analysis to study the effect of a model or log characteristic on algorithm performance. The validition consists of an experiment that empirically analyzes four process discovery algorithms that apply different target modelling notations: Alpha+ Miner [106] (Petri nets), Heuristics Miner [128] (Heuristic nets), ILP Miner [114] (Petri nets) and the Inductive Miner [67] (Process trees). The discovery algorithms come from three different "families" of approaches (see Section 2.1.4): two algorithmic approaches (Alpha+ and Inductive), a heuristics approach (Heuristics miner), and a region-based approach (ILP miner). The benchmark does not serve as a benchmark of all current state-of-the-art algorithms. The chosen algorithms are are not the most recent approaches, yet, more knowledge has accumulated for these algorithms than previous algorithms, which benefits the evaluation for theory building and validation of those theories. The experiment applies the proposed procedure to analyze and compare the quality of process discovery algorithms to rediscover the underlying process based on observed executions (event logs). The process to be rediscovered can come from different populations by varying the probability of occurrences of typical process characteristics such as paralellism, loops and infrequent paths. The experiment in this round will test discovery algorithms on event logs coming from different model populations by varying the probability of reoccurring activities and enabling or disabling the presence of infrequent paths. In this way, we can study the impact of infrequent behavior and of different probabilities of reoccurring activities on the quality of process discovery techniques.

2. The second round will further validate the proposed procedure by showing the flexibility and its support for large-scale experiments. It adapts the first experiment to an experiment four times as large.

First, we will discuss the setup of the first round of experiments, followed by a description of the results. Secondly, we will present the setup of the second round of experiments and its results. Finally, we will discuss the results from both

Discovery Technique	**Infrequent Paths**	**Probability Reoccurring Activities**
Alpha+ [106]	False	0.0, 0.05
Heuristics [128]	True	0.10, 0.15
ILP [114]		0.20, 0.25
Inductive [67]		0.30

Table 4.3: Summary of the possible values of the four variables included in the experimental setup: 56 (4 × 2 × 7) value combinations. The probability of reoccurring activities indicates the average percentage of duplicated visible activity labels in the process model.

rounds of experiments and how the presented procedure alleviates important limitations of current evaluation procedures.

4.3.1 Setup of the first round of experiments

The first experiment compares the quality of the four process discovery techniques to rediscover the underlying process, i.e. to perform a benchmark analysis. Furthermore, it analyzes the impact of infrequent behavior and of different probabilities of reoccurring activities on the quality of process discovery techniques to rediscover the underlying process, i.e. a sensitivity analysis. Therefore, the experimental design includes all the combinations of three independent variables: process discovery technique used, presence or absence of infrequent paths and the probability of having reoccurring activities. The three variables and their levels are summarized in Table 4.3. In total, the 56 possible combinations are included in the experiment: 4 discovery techniques × 2 levels of infrequent behavior × 7 probabilities of reoccurring activities.

The discovery algorithms are applied with their default parameter settings, except for the ILP miner which is set to discover Petri nets where the final marking is the empty marking (no tokens remaining). Any other configuration results in Petri nets that would require manual inspection by the researcher to determine the final marking(s) for the replay during the conformance checking. However, a manual intervention would hinder the automatic execution of the evaluation procedure.

As mentioned above, we vary the presence/absence of infrequent paths and the probability of reoccurring tasks as these are part of the independent variables. The other process characteristics are fixed for each model population. More specifically, the probabilities of the "loop" and "or" control-flow patterns are set to zero, and hence, these patterns do not occur in the model population. The probability of the "sequence", "exclusive choice" and "parallelism" patterns is set and kept fixed to values 46%, 35% and 19%, respectively. These values have been determined after analysing their frequencies in the large collections of models reported in [65]. In this work, Kunze et al. have observed that 95% of the models consist of activities connected in sequences, 70% of the models consist of activities, sequences and XOR (exclusive choice) connectors and 38% consist of sequences, activities and AND (parallel) connectors (see Figure 4b of the paper). Assuming independence of occurrence probability of sequences, AND and XOR, it follows that:

$$P(sequence) = 0.95$$
$$P(sequence \wedge XOR) = P(sequence) \times P(XOR) = 0.70 \Rightarrow P(XOR) = 0.74$$
$$P(sequence \wedge AND) = P(sequence) \times P(AND) = 0.38 \Rightarrow P(AND) = 0.4$$

When these values are normalized to 1, the final probabilities of 46%, 35% and 19% for the "sequence", "exclusive choice" and "parallelism" patterns are obtained. The size of the models within each population varies between 15 and 60 with a mode of 30. This makes the 14 model population definitions MP_{first} as shown in Table 4.4 where X and Y are assigned all 14 combinations of values in column two and three in Table 4.3.[6] As such the 14 model populations contain mostly sequential process models that have mutually exclusive choices and to a lesser extent contain some parallel behavior. The size of the models in terms of visible activities has a relatively large range to include both simple and rather complex models in terms of nested choices and parallellism. Furthermore, half of the models produce infrequent traces in the log because all their choices have imbalanced outgoing branch probabilities: 90% probability to execute one

[6] $\Pi^{In} = 0$ for "False" (absence of infrequent paths) and $\Pi^{In} = 1$ for "True" (presence of infrequent paths).

Parameter	**Population $\mathbf{MP_{first}}$**
Number of visible activities	(15,30,60)
Sequence ($\Pi^{\rightarrow}$)	0.46
Parallel ($\Pi^{\wedge}$)	0.19
Choice ($\Pi^{\times}$)	0.35
Loop ($\Pi^{\circlearrowleft}$)	0
Or ($\Pi^{\vee}$)	0
Silent activities (Π^{τ})	0
Reoccurring activities (Π^{Re})	X
Long-term dependencies (Π^{Lt})	0
Unfold loops	n/a
Max repeat (k)	n/a
Infrequent paths (Π^{In})	Y
Sample size (number of trees)	62
Logs per model	1
Number of traces (t)	[200,1000]
Noise (Π^{Noise})	0
Determinism level	n/a

Table 4.4: Model population parameters for the first round of experiments, where X and Y are assigned all 14 combinations of values in {0,0.05,0.1,0.15,0.2,0.25,0.3} and {0 (False),1 (True)} respectively.

branch, and 10% probability to execute one of the other branches. Finally, the models are characterized by different amounts of reoccurring activities, ranging from no reoccurring activities to models with on average 30% of all visible activities taking on labels of other visible activities.

As mentioned in Section 4.1.1 the evaluation procedure ensures that every discovery algorithm is tested on a different event log to make all discovered models and quality metrics for a specific algorithm independent observations. Therefore, for each discovery technique, a random sample of 62 process models is drawn. The sample size of 62 models allows us to study the effect of process discovery techniques, infrequent paths and different probabilities of duplicate-activity occurrences (and their interactions) using a fixed effects ANOVA analysis [122] with significance level $\alpha = 0.05$ and power $1 - \beta = 0.98$.[7] This power indicates the probability to detect a significant effect when two mining algorithms actually differ by a relatively small difference. The small difference is defined by an effect size equal to 0.1, i.e. a standard deviation of the effects we want to test is one-tenth as large as the common standard deviation of the observations within the groups [23]. This experiment tests three effects: discovery algorithm, infrequent paths and reoccurring activities. The number of groups equals 56, i.e. the number of combinations of the three independent variables: 4 discovery algorithms x 2 levels of infrequent paths x 7 levels of reoccurring activities. In total 3472 process models were generated: 56 groups x 62 models = 3472 models.

For each of the obtained process models, an event log containing between 200 and 1000 traces is generated. For each generated log, we can calculate the completeness, i.e. the ratio of unique traces in the log to all possible unique traces according to the model using the technique described in [49]. Fig. 4.2 shows that the completeness of the logs varies between 0 and the maximum of 1.

Finally, the non-fitting traces in the test logs are generated using the following noise operations: remove one activity from a trace, duplicate an activity, and swap consecutive activities. Each of the noise operations has an equal probability to be applied. Notice, that the applied noise operations and their probabilities should be taken into account when generalizing the experiment results.

The setup of the experiment with all the parameter settings in a RapidProM workflow is shared via the following url: `http://data.4tu.nl/repository/uuid:039bc213-b09d-4c96-bc1d-15bc13f645e6`.

4.3.2 Analysis of the results from the first experiment

The effect of process-discovery techniques, infrequent paths and different probabilities of duplicate-activity occurrences can be analyzed using one-way ANOVA analysis if the assumptions of homogeneity of variances and normality of the dependent variable hold [122]. However, both assumptions were violated for every

[7]The power was computed with the G*Power tool [38].

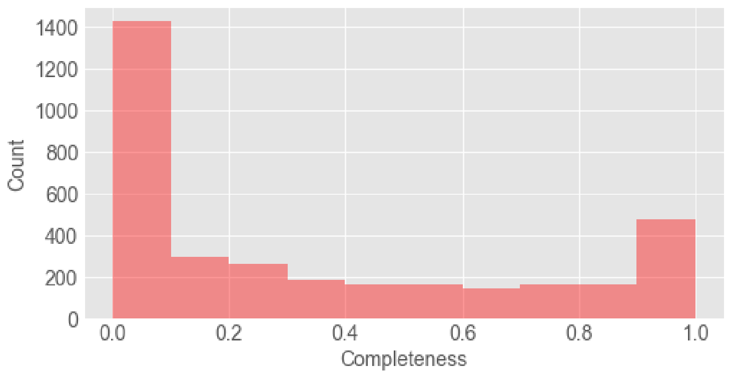


Figure 4.2: Distribution of completeness of logs wrt. their respective process models. Completeness is measured as the fraction of traces allowed by the model that are present in the event log.

dependent variable, i.e. F_1, recall and precision. Therefore, the non-parametric *Kruskall-Wallis* test (KW) [94] was applied instead. We will first introduce the statistical tests used during analysis, before we discuss the analysis results.

4.3.2.1 Statistical tests

KW is used for testing whether k independent samples are from different populations. It starts by ranking all the observations from the different samples together based on their scores: assign the highest score a rank 1 and the lowest a rank N, where N is the total number of observations in the k samples. Then, the average ranking for each sample is computed, e.g. the mean ranking of observations in sample j is denoted as $\bar{R}_j$. With n the number of observations in each sample, the test statistic KW, which follows a χ^2 distribution with $k-1$ degrees of freedom, can be calculated as follows [94]:

$$\mathrm{KW} = \left[\frac{12}{(N(N+1))} \sum_{j=1}^{k} n\bar{R}_j^2 \right] - 3(N+1)$$

If the calculated KW is significant, then it indicates that at least one of the samples is different from at least one of the others. Subsequently, the multiple comparison post hoc test is applied to determine which samples are different.

More specifically, for all a pairs of samples R_i and R_j it is tested whether they differ significantly from each other using the inequality [94]:

$$|R_i - R_j| \geq z_{\alpha/k(k-1)} \sqrt{\frac{N(N+1)}{12}\left(\frac{2}{n}\right)}$$

The $z_{\alpha/k(k-1)}$ value can be obtained from a normal distribution table given a significance level α. The formula adjusts this α with a Bonferroni correction to compensate for multiple comparisons. If the absolute value of the difference in average ranks is greater than or equal to the critical value, i.e. the right side of the equation, then the difference is significant.

Finally, the Jonckheere test [94] can be used to test for a significant trend between the k samples. First, arrange the samples according to the hypothesized trend, e.g. in case of a positive trend from smallest hypothesized mean to largest hypothesized mean. Then count the number of times an observation in sample i precedes an observation in sample j, denoted as U_{ij} $\forall i < j$. The Jonckheere test statistic J is the total number of these counts:

$$\mathrm{J} = \sum_{i<j}^{k} U_{ij}$$

When J is greater than the critical value (see [94] for the sampling distribution) for a given significance level α, then the trend between the k samples is significant.

4.3.2.2 The effect of process discovery technique

The goal is to learn the effect of a process discovery technique on each of the dependent variables: recall, precision and F_1 score. The effects of the other independent variables, i.e. infrequent paths level and probability of reoccurring activities, are not studied here.

We apply the KW method, to test whether the average rank differs between the four process discovery techniques (i.e. samples). In this case we ranked all the 3472 averages over the 10-fold cross validation for recall, precision and F_1 values ignoring sample membership (i.e. discovery technique). The highest value for recall, precision and F_1 gets rank 1 (lowest rank), while the lowest absolute

	Alpha+	**Heuristics**	**ILP**	**Inductive**
Recall	2361.94	2650.35	505.99	1427.73
Precision	2155.57	2624.42	1007.66	1158.35
$\mathbf{F_1}$ score	2318.14	2646.44	697.00	1284.42

Table 4.5: Average Ranks per Miner. Each cell indicates the average ranking for a specific quality dimension (row header) and for a specific miner (column header). One can compare miners by comparing the average ranks within one row.

value gets rank 3472 (highest rank). Then we computed the average ranking per miner, i.e. the average position of a discovered model by that miner for that quality metric on a scale from 1 to 3472. A higher average ranking means worse performance. The ranking summary is shown in Table 4.5.

Based on the average rankings in Table 4.5, the order suggested between process discovery techniques is: ILP > Inductive > Alpha+ > Heuristics for recall, precision and F_1 scores. It means that the ILP miner creates the best models in terms of recall, precision and F_1 scores (see Section 4.3.4 for an elaborate discussion). The Inductive miner outperforms the Alpha+ miner, which in turn outperforms the Heuristics miner. The results of the KW test confirm that the differences in average rankings between the four miners are statistically significant (significance level $\alpha = 0.05$). Moreover, the multiple comparison post-hoc test (cf. supra) also confirms the statistical significance of the differences between algorithms. See Table B.1 in Appendix B for a summary of the statistical test results for the F_1 scores.

4.3.2.3 The effect of infrequent paths

This analysis tests whether the presence/absence of infrequent paths has an impact on the average ranking of the four process discovery techniques for recall, precision and F_1 scores.[8] The effect of reoccurring activities is not studied here.

[8]Infrequent paths are denoted with an imbalance in execution probabilities of the output-branches of each exclusive choice construct in the model which results in an event log containing infrequent behavior.

	Alpha+	**Heuristics**	**ILP**	**Inductive**
Recall	1162.64	1313.58	242.94	754.85
Precision	1088.57	1306.34	523.05	556.04
F_1 score	1136.22	1310.40	367.71	659.93

Table 4.6: Average ranks per miner with infrequent behavior

	Alpha+	**Heuristics**	**ILP**	**Inductive**
Recall	1196.47	1338.58	262.01	676.93
Precision	1063.51	1317.79	485.44	607.26
F_1 score	1180.24	1338.25	325.71	629.80

Table 4.7: Average Ranks per miner without infrequent behavior

Firstly, the sample is split into two subsets: experiments with infrequent behavior and experiments without infrequent behavior. This division is called *blocking* (see Section 2.3.2.2) which is done to isolate the variation in recall, precision and F_1 scores attributable to the absence/presence of infrequent paths. Secondly, the KW test is applied to each subset.

Tables 4.6 and 4.7 contain the average rankings per process discovery technique grouped by metric and experiments with and without infrequent behavior respectively. These rankings suggest the same order between process discovery techniques in all cases: ILP > Inductive > Alpha+ > Heuristics. This leads to the assumption that the process discovery techniques are not influenced by the absence or presence of infrequent behavior. Based on the KW and multiple comparison post-hoc test, only the difference between the ILP and Inductive miner in case of infrequent behavior is not statistically significant for precision (see Table B.2 in Appendix B), i.e. the rankings are ILP ? Inductive > Alpha+ > Heuristics. Therefore, the rankings between algorithms are not always ILP > Inductive > Alpha+ > Heuristics for all quality dimensions for both with and without infrequent behavior. As a result, one cannot accept the assumption that infrequent paths do not influence process discovery techniques.

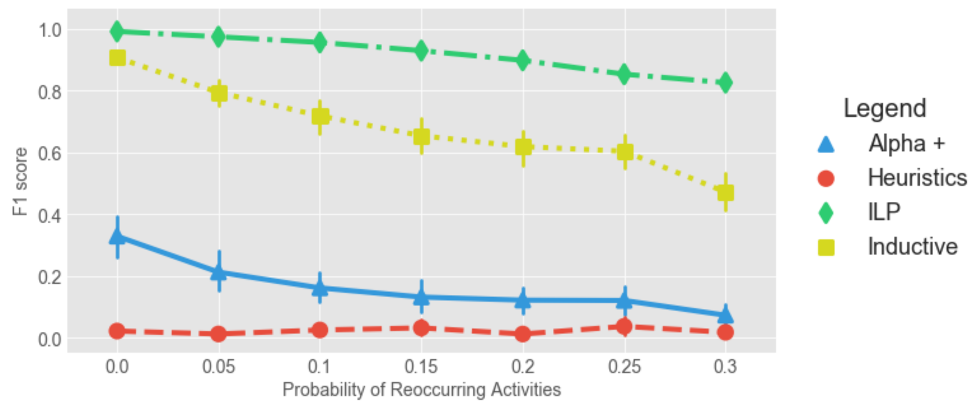


Figure 4.3: F_1 scores for process discovery techniques for different probabilities of Reoccurring activities

4.3.2.4 The effect of reoccurring activities

This analysis investigates how the quality of each process discovery technique (in terms of precision, recall and F_1 score) is influenced by the probability of reoccurring activities (i.e. the average percentage of duplicated visible activity labels in the process models). The effect of infrequent behavior is not studied here.

Fig. 4.3 illustrates the average F_1 scores for all the process discovery techniques over different probabilities of reoccurring activities. We have added error bars to the means displayed in the graphs. The bars are based on a 95% confidence interval. These bars show that only the Alpha+ and Inductive miner have some variability. The graph indicates a negative trend, i.e. the probability of reoccurring activities has a negative effect on F_1 scores. To determine whether such a trend is statistically significant, an in-depth analysis is performed.

First, the sample is divided into subsets grouped by process discovery technique. As such, the variation in accuracy associated with the discovery technique is isolated. Then, similar to the analysis above, the KW test is applied to compare the average rankings of the discovered models.

Tables 4.8, 4.9, 4.10, and 4.11 contain the average ranks of the four discovery algorithms for all three metrics per probability of reoccurring activities.

Prob. Reoccurring Activities	0	0.05	0.10	0.15	0.20	0.25	0.30
Recall	339.47	417.21	428.08	451.02	435.54	479.52	490.66
Precision	346.37	421.59	423.21	449.60	431.10	478.01	491.62
F_1 score	339.46	417.56	427.86	450.78	435.69	479.36	490.78

Table 4.8: Average ranks of Alpha+ miner per probability of Reoccurring Activities

Prob. Reoccurring Activities	0	0.05	0.10	0.15	0.20	0.25	0.30
Recall	432.08	429.27	435.26	428.28	442.85	437.97	435.79
Precision	431.96	429.40	434.90	428.14	442.76	438.51	435.83
F_1 score	432.04	429.34	435.19	428.27	442.85	438.06	435.75

Table 4.9: Average ranks of Heuristics Miner per probability of Reoccurring Activities

Prob. Reoccurring Activities	0	0.05	0.10	0.15	0.20	0.25	0.30
Recall	466.09	438.65	457.05	462.85	416.74	418.97	381.15
Precision	172.53	295.21	369.48	431.76	521.84	604.08	646.60
F_1 score	185.64	288.92	370.51	430.34	519.42	602.04	644.64

Table 4.10: Average ranks of ILP miner per probability of Reoccurring Activities

Prob. Reoccurring Activities	0	0.05	0.10	0.15	0.20	0.25	0.30
Recall	239.90	339.92	415.75	462.62	492.83	515.94	574.54
Precision	241.90	331.42	405.01	478.28	496.85	490.82	597.22
F_1 score	219.71	339.45	410.69	474.73	500.88	505.77	590.27

Table 4.11: Average ranks of Inductive miner per probability of Reoccurring Activities

For the Alpha+ Miner, the data (shown in Table 4.8) seems to suggest that as the probability of reoccurring activities increases, the models generated by Alpha+ miner deteriorate in terms of recall, precision and F_1 score. To test this impression statistically, we will rely on the KW and Jonckheere tests. Both tests show that there is statistically significant negative trend in the relative quality of the generated models as the probability of reoccurring activities increases. A pairwise comparison of each probability of reoccurring activities does not provide a clear picture how this trend looks like for recall, with many comparisons statistically insignificant. For precision and F_1 on the other hand, the quality of the models decreases significantly whenever the probability of reoccurring activities increases from 0% to more than or equal to 15% (see Table B.3 in Appendix B).

The models discovered using the Heuristics Miner seem insensitive to the probability of reoccurring activities (see Table 4.9). The KW and Jonckheere tests confirm that there is indeed statistically insufficient evidence of a trend in recall, precision and F_1 score as the probability of reoccurring activities increases (see Table B.4 in Appendix B). A possible explanation will be discussed in Section 4.3.4.

The results for the ILP Miner in Table 4.10 suggest a positive trend in the probability of reoccurring activities in terms of recall! However, in terms of precision, the ILP miner shows high sensitivity to the probability of reoccurring activities. The KW and Jonckheere tests confirm both statements. The pairwise comparisons of reoccurring activities reveals the significant negative trend in terms of precision and F_1 scores of the generated models as the probability of reoccurring activities increases (see Table B.5 in Appendix B).

The findings for the Inductive Miner in Table 4.11 indicate that as the probability of reoccurring activities increases, the model quality in terms of recall, precision and F_1 score deteriorates. This effect, though, seems to level off as we reach higher probabilities of reoccurring activities. The KW and Jonckheere tests show that there is indeed a significant negative trend in the relative quality of the generated models as the probability of reoccurring activities increases. However, at a probability of around 15% of reoccurring activities, this effect seems to have reached a plateau and stays stable (see Table B.6 in Appendix B).

4.3.3 Second experiment

The first experiment has validated that the proposed evaluation procedure supports the benchmark and sensitivity analysis evaluation objectives. The proposed procedure is also flexible as it allows users to easily set up extended experiments. Here, we have extended the above experiment with four other control-flow patterns that provide challenges to current discovery algorithms: "loop", "or", "silent activities", and "long-term dependencies". The probability of the basic patterns, "sequence", "parallel" and "exclusive choice", is set the same as in the previous experiments. In this experiment we have varied the probability of each of the four patterns occurrence in the same way as we varied the probability of "reoccurring activities" in the first experiment (see the model population definitions in Table 4.12). Additionally, also the absence/presence of infrequent paths is varied in the same way as in the first experiment. As a result, the extended experiment has 4 x 3472 = 13888 observations.

The graphs in Figures 4.4, 4.5, 4.6, and 4.7 show the average F_1 score for all the discovery techniques over different probabilities of inclusion of the control-flow patterns. We have added error bars to the means displayed in the graphs. The bars are based on a 95% confidence interval. Overall the Alpha+ miner displays the largest variability while error bars of other miners are very small (and thus barely visible). One can apply a similar statistical analysis of the experiment results as for the Reoccurring activities to assess, for example, whether the negative trend for the ILP miner on "or" is statistically significant. The next subsection details a thorough discussion of the results of this extended experiment, along with the first experiment.

4.3.4 Discussion of results

Firstly, the graphs on how the different algorithms score in terms of F_1 score (see Figures 4.3, 4.4, 4.5, 4.6, and 4.7) clearly highlight that ILP and Inductive Miner perform significantly better than Alpha+ and Heuristic Miner. In fact, this is not surprising because the latter two miners are not guaranteed to produce sound models, which allow executions to be carried out till completion. Models discovered with Alpha+ and Heuristic Miner can contain deadlocks, livelocks,

Parameter	**Population $\mathbf{MP}_{\circlearrowleft}$**	**Population $\mathbf{MP}_{\vee}$**	**Population $\mathbf{MP}_{\tau}$**	**Population $\mathbf{MP}_{\mathbf{ltd}}$**
Number of visible activities	(15,30,60)	(15,30,60)	(15,30,60)	(15,30,60)
Sequence ($\Pi^{\rightarrow}$)	0.46	0.46	0.46	0.46
Parallel ($\Pi^{\wedge}$)	0.19	0.19	0.19	0.19
Choice ($\Pi^{\times}$)	0.35	0.35	0.35	0.35
Loop ($\Pi^{\circlearrowleft}$)	**X**	0	0	0
Or ($\Pi^{\vee}$)	0	**X**	0	0
Silent activities (Π^{τ})	0	0	**X**	0
Reoccurring activities (Π^{Re})	0	0	0	0
Long-term dependencies (Π^{Lt})	0	0	0	**X**
Unfold loops	n/a	n/a	n/a	True
Max repeat (k)	n/a	n/a	n/a	1
Infrequent paths (Π^{In})	Y	Y	Y	Y
Sample size (number of trees)	62	62	62	62
Logs per model	1	1	1	1
Number of traces (t)	[200,1000]	[200,1000]	[200,1000]	[200,1000]
Noise (Π^{Noise})	0	0	0	0
Determinism level	n/a	n/a	n/a	n/a

Table 4.12: Model population parameters for the second round of experiments, where X and Y are assigned all 14 combinations of values in {0,0.05,0.1,0.15,0.2,0.25,0.3} and {0 (False),1 (True)} respectively.

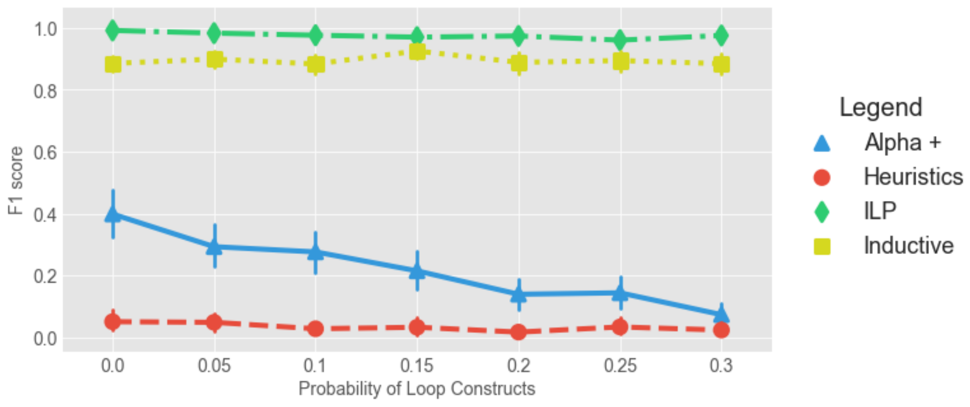


Figure 4.4: F_1 scores for process discovery techniques for different probabilities of Loops

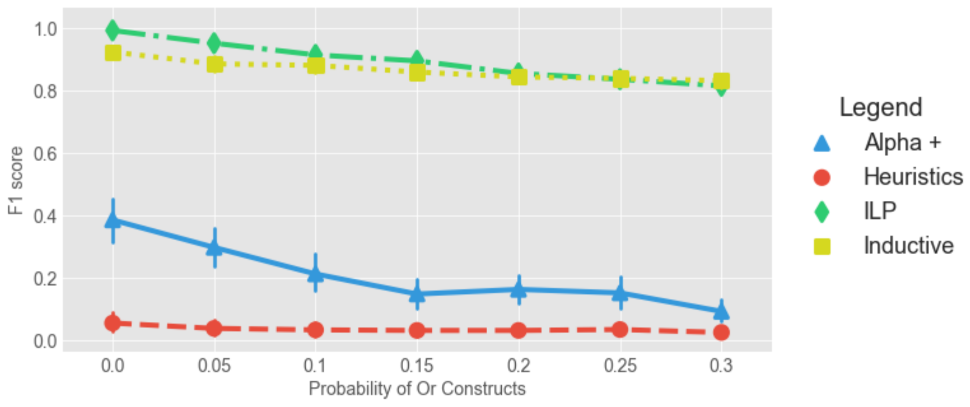


Figure 4.5: F_1 scores for process discovery techniques for different probabilities of OR

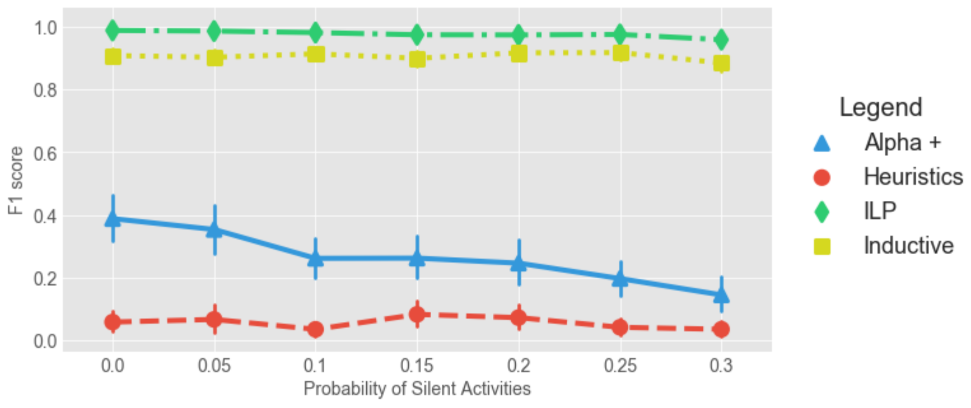


Figure 4.6: F_1 scores for process discovery techniques for different probabilities of Silent Transitions

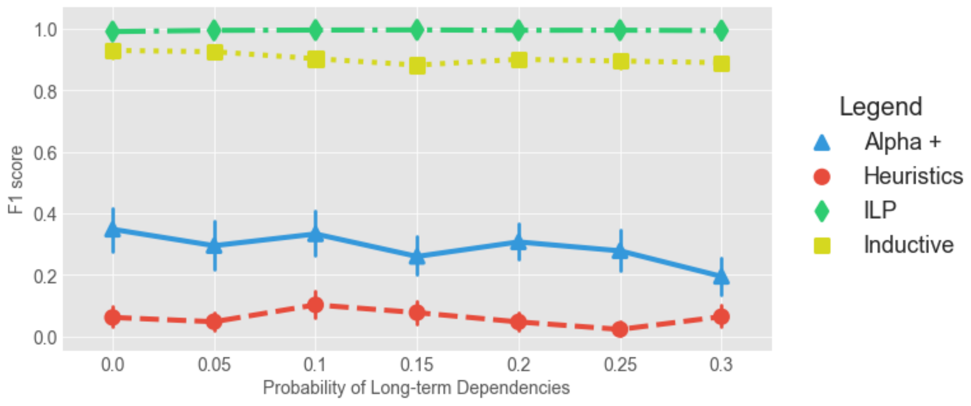


Figure 4.7: F_1 scores for process discovery techniques for different probabilities of Long-term Dependencies

and other anomalies [99]. When a model is not sound, it cannot replay traces until the end and, hence, the confusion matrix may contain few true positives (often none), causing precision, recall and, hence, F_1 scores to be very low (often zero).

Figures 4.8 and 4.9 show unsound models discovered by the Alpha+ Miner and Heuristics Miner respectively. The model discovered by the Alpha+ Miner immediately deadlocks after executing one of the following activities as first activity in the process: o, h, or q. Additionally, the process model does not end properly as the end marking can have multiple tokens. The model discovered by the Heuristics Miner deadlocks after executing the silent activities indicated by the red boxes. For both models such unsound behavior results in all zero scores for recall, precision and F_1.

The very low quality scores for the Alpha+ and Heuristics miners are not a trivial findings because, although the theory already postulated it, it was not clear how much the lack of soundness guarantee was practically affecting the results. Ultimately this means that Alpha+ Miner and Heuristic Miner can be useful to gain an initial insight into the general structure of the process but cannot be used for more mature answers or for automatically generating models that can be plugged into a Process-Aware Information System to enforce certain process behavior.

The charts indicate that the ILP miner tends to perform better than Inductive Miner in terms of F_1 score. This is observed for all patterns and all occurrence probabilities. In particular, for such patterns as silent activities and long-term dependencies, the F_1 score is steadily around 1, which indicates almost perfect precision and recall. This result is far from being trivial: as discussed in [114], the ILP miner focuses on producing models that can replay every trace of the event log, without trying to maximize precision. The ILP miner starts from a Petri net that only contains transitions and add places to restrict the possible behavior. It stops searching for places from the moment all causal dependencies between transitions as found in the log are expressed. In case there is exceptional behavior in the log, then ILP produces a WF-net that allows for all this exceptional behavior. As a result, the ILP miner tends to not find any causal dependencies. This typically leads to places with self-loops

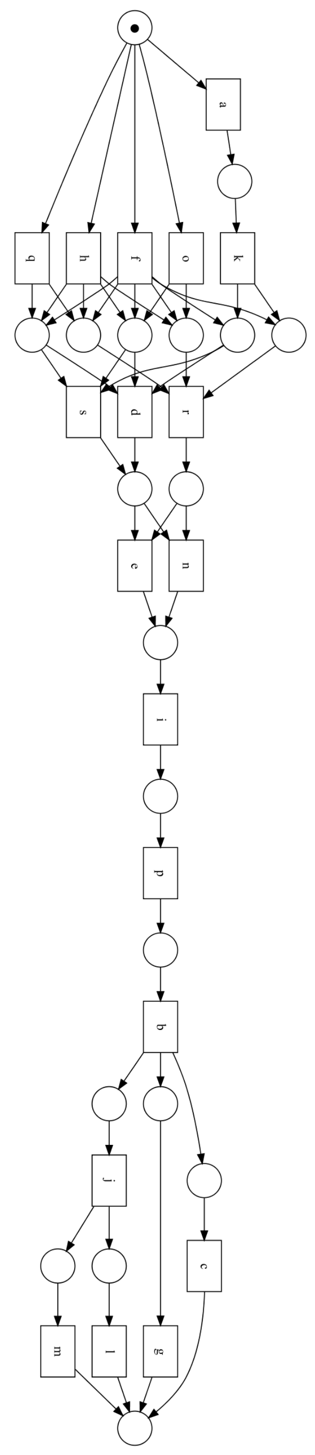


Figure 4.8: Model discovered by the Alpha+ Miner.

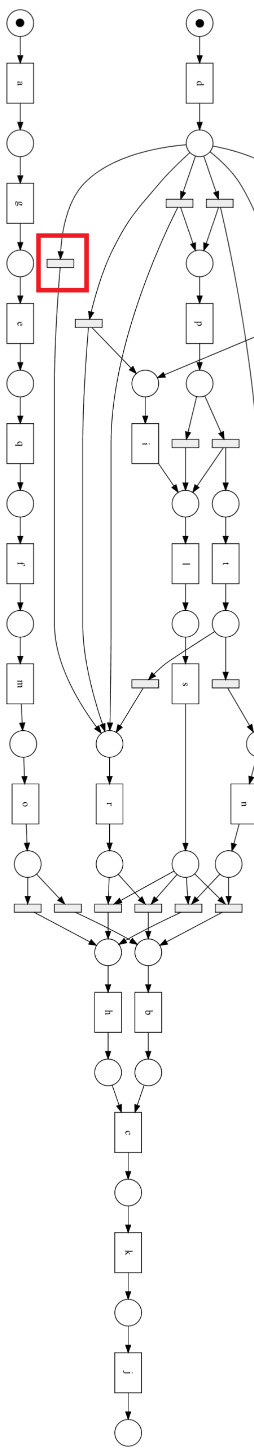


Figure 4.9: Model discovered by the Heuristics Miner.

and as such underfitting models in general. In the experiments, the presence of infrequent paths (exceptional behavior) still resulted in high precision scores. Furthermore, because the ILP miner only aims at replaying the traces in the event log used for discovery, one would expect that a different event log, such as a test log, would not let the discovered models score high in recall, either.

The superiority of ILP miner is further supported by visually comparing the models that ILP generates and those from the Inductive Miner, such as the models in Figures 4.10 and 4.11 respectively discovered by the Inductive and ILP Miner. The red boxes in the figure illustrate the imprecise parts of the model. For the Inductive-Miner model, the transitions in the box can be executed in any order and, because of the loop, an arbitrary number of times. Of course, in reality, these transitions should occur in a more precise order; but the miner is unable to discover it. Conversely, for the model discovered by the ILP miner, the only "source of imprecision" is related to the "floating transition" a but it is just one out of 26 transitions. This does not affect the precision. As discussed in Section 4.1.2.4, to punish for imprecise behavior, our procedure injects noise into fitting traces. In case of the model by the ILP miner, the probability that the noise would involve the only "floating transition" a is low. On the other hand, the probability that noise affects activities present in precise regions of the model is high. Such deviations in very precise regions are easily detected, resulting in high F_1 scores for the ILP miner. The same reasoning is shared among most of models illustrating the superiority of the ILP miner.

Another interesting result for both Inductive and ILP miner is that the values of F_1 score do not seem to be really affected by the amount of occurrences of the "loop", "silent activities", and "long-term dependencies", except for "reoccurring activities" and, limitedly, from the "or" pattern. The "or" is known to be a hard construct and neither of the two miners provides specific support for it (for Inductive Miner, at least for the version being evaluated[9]). For reoccurring activities, this can be explained by the fact that both ILP and Inductive Miner do not natively support mining models where different transitions share the same activity label. This means that reoccurring activities are "emulated" through

[9]We have applied the Inductive Miner - infrequent variant, while alternatively, one could also apply the Inductive Miner - infrequent - all operators variant to discover "or" patterns [68].

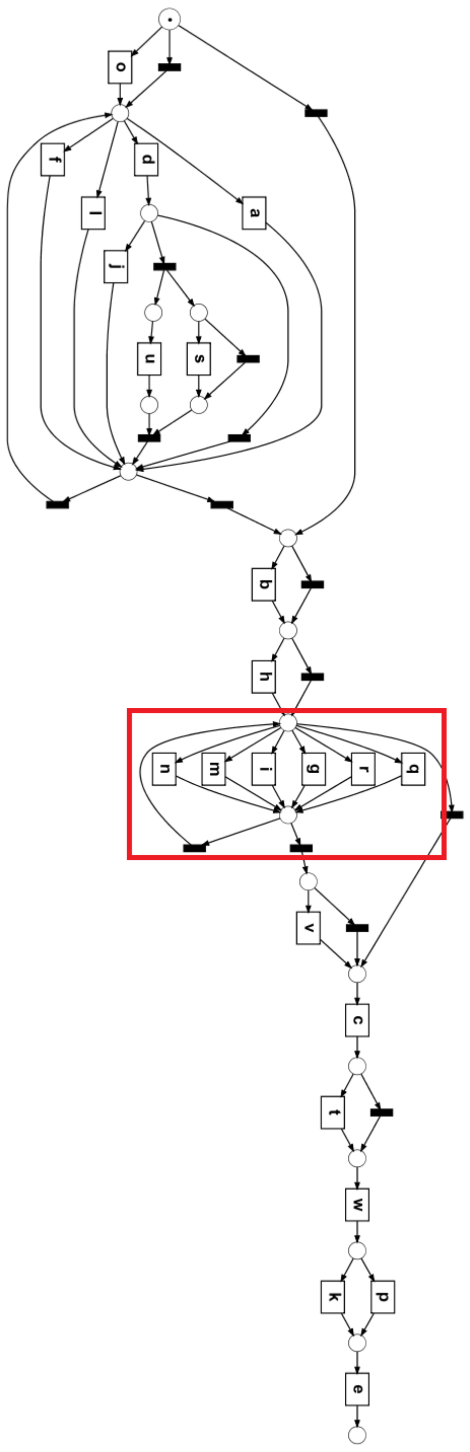


Figure 4.10: Model Discovered by the Inductive Miner. The red box highlights the imprecise part of the model.

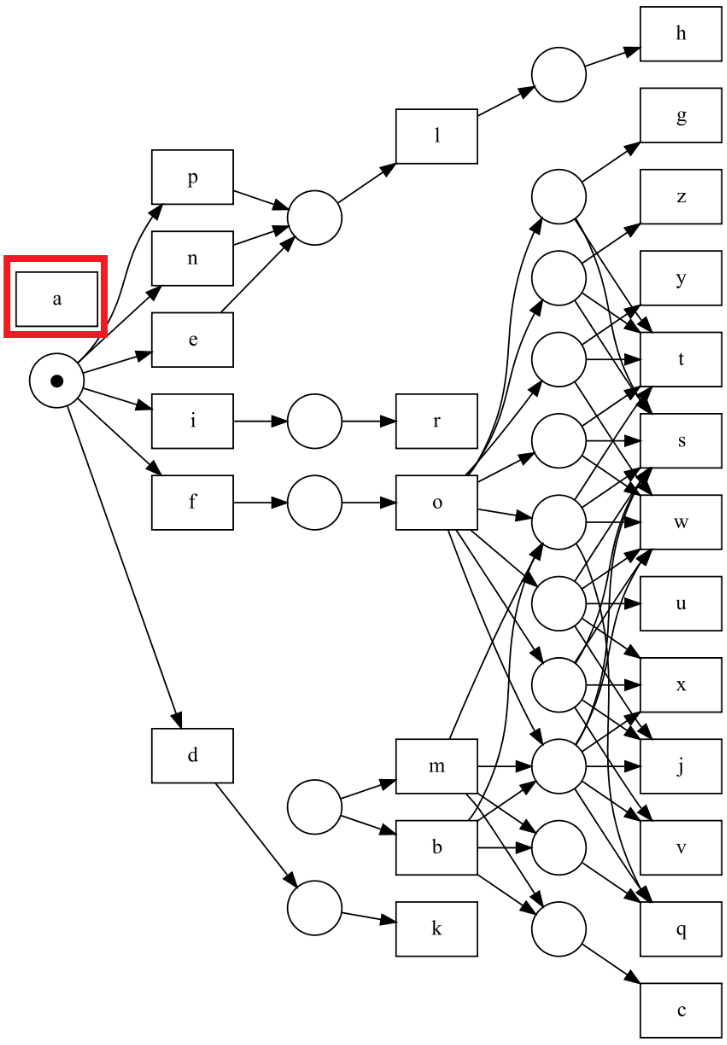


Figure 4.11: Model Discovered by the ILP Miner. The red box highlights the imprecise part of the model.

loops and floating transitions (see above), which would underfit the behavior observed in the event log, thereby yielding low precision.

4.3.5 Limitations and threats to validity

This subsection will discuss the limitations of the procedure's implementation and performed experiments. The first limitation regards the lack of noise in the training logs and the way noise is inserted to create non-fitting traces. The second limitation concerns the data preparation and discovery algorithm's parameter optimization. The final limitation involves the required conversion to Petri nets for the conformance checking.

4.3.5.1 Noise

We acknowledge that the experiment results are affected by the fact that training event logs do not contain noise, namely traces that are not generated by the original, artificial models. As an example, ILP miner tends to be very sensitive to noise: since it discovers models that are able to replay every trace, if the logs contain noise, the discovered models would incorporate behavior that should not be allowed, thus negatively affecting precision. Conversely, Inductive Miner would likely be less affected because it features some noise detection, able to detect whether a trace is really part of the process or a noise/outlier. This is based on the frequencies of occurrences of certain patterns in the traces of the event log [67]. As future work, we aim to add new ingredients to our analysis and consider a variable percentage of training-log noise and to study how discovery algorithms are affected by the amount of noise, in terms of F_1 score.

Furthermore, we acknowledge that the subset of noise operators, i.e. remove one activity from a trace, duplicate an activity, and swap consecutive activities, has affected the experiment results. This subset was tailored towards testing whether discovery algorithms introduce activity skips, loops and parallelism that make the discovered model imprecise. However, there could be other sources of imprecisions in the discovered models as well that cannot be detected using our subset of noise operations. Consider for example the model in Figure 4.12 where the grey places express a long-term dependency between activities a and e and

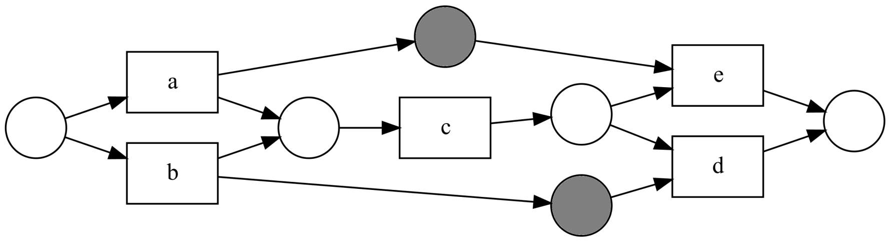


Figure 4.12: An example of a long-term dependency.

b and d such that only the traces $\langle a, c, e \rangle$ and $\langle b, c, d \rangle$ are allowed by the original model. Both the Inductive Miner and the ILP Miner will discover an underfitting model without the grey places and therefore also allow for the traces $\langle a, c, d \rangle$ and $\langle b, c, e \rangle$. The noise operators duplicate, swap and remove can never generate these additional traces, and therefore, cannot punish discovered models in terms of precision. An alternative way to generate traces that can punish imprecision with regards to long-term dependencies collects all the removed branches from the unfolded choice tree during dependency insertion (see Section 3.2.4). These removed branches can be used to generate traces that do not fit the "orginal model" with long-term dependencies, e.g. trace $\langle a, c, d \rangle$ which would lower the precision score for imprecise models such as the model without the grey places in Figure 4.12. Future research on process discovery evaluation should look into such additional ways to generate unfitting test traces to detect all imprecisions that current state-of-the-art discovery techniques tend to produce. Additionally, future research should investigate how the random way of introducing noise could be replaced by an approach guided by the original model. The original model can be used to steer the non-fitting behavior in such a way that activities are removed, duplicated, or swapped in "realistic" places. The resulting non-fitting traces are likely to be more difficult for discovery algorithms to classify correctly as non-fitting.

4.3.5.2 Parameter optimization and data preparation

A future extension to our procedure is parameter sensitivity [18, 86]. Every miner that we employed in our experiments can be customized by setting the values of certain parameters. In this thesis, we ruled out the parameter sensitivity by using the default parameter values. For instance, Inductive Miner can be customized by varying the threshold of noise detection, also known in the algorithm as α-value, which can vary from zero to one. The model in Fig. 4.10 was mined with the default α-value, which is 0.2, leading to a F_1 score of 0.25. For this specific case, we manually reduced α to 0, thus not supporting noise detection. This led to an increase of F_1 till a clearly better 0.67. The increase was caused by the fact that the training logs do not contain noise which benefits a lower α setting.

Next to the parameters, also the assumptions of a discovery algorithm could impact the quality of the discovered models. The Heuristics miner assumes that each process has a unique start and end activity. However, this assumption is not guaranteed in the experiments, as the original model could have more than one start or end activity (due to choices or concurrency). Consequently, this partly explains the bad results of the Heuristics miner in the experiments. To illustrate this, consider the model in Figure 4.14. The initial marking is in the place before transition “a” such that the process can only start with transition “a”, while according to the original model the process can start with transitions “a”, “l”, “z” or “r”. If a test trace starts with activity “z” for example, then that trace will be classified by the discovered model as not-completely replayable. One could alleviate this last assumption by introducing a dummy start and end activity for each trace in the model. However, the current experiments in the thesis are explorative which justifies the adoption of default parameters without preprocessing such as introducing those artificial start and end activities.

4.3.5.3 Impact of conversion on model soundness

The use of alignment-based trace fitness can lead to bad quality results in case the discovery algorithm does not guarantee to return sound models. The previous section (see Section 4.3.4) explained that the lack of soundness for the Alpha+ and Heuristics miners leads to relatively bad quality results. Given the theoretical underpinnings of the Heuristics miner, these results were unexpected. Therefore, this paragraph further analyzes the results of the Heuristics miner.

The first problem arises because of the required conversion. The heuristics miner produces heuristics nets (causal matrices) which are slightly more expressive than classical Petri nets [76]. The translation of a Heuristics net to a Petri net requires some silent transitions which are the cause of local unsoundness due to improper completion and/or deadlocks.

A second problem arises because the converted unsound Petri net influences the computation of alignments. The assumption of alignments with regard to the input process model is as follows [71]:

The Petri net should be relaxed sound: there needs to be at least one sequence of transition firings that leads from the initial state to a final state. This restriction is motivated by the fact that the process projection of an alignment is required to be a process trace of the model. Clearly, to be able to provide an alignment, the Petri net should allow for at least one trace. This does not require Petri nets without deadlocks, i.e. states different from the final state in which no transitions are enabled, or livelocks, i.e. firing sequences that do not progress towards the final state.

The converted models are not necessarily relaxed sound, i.e. there is no guarantee that at least one sequence of transition firings exists that leads from the initial state to a final state. To illustrate this, several models and fitting test traces where analyzed. The models and test traces are illustrated on the following pages. Below each figure there is a table with a test trace and the activities that are replayable and non-replayable doing a manual token-based replay.

Firstly, a model that could not be aligned to the test log is analyzed. The original Heuristics net in (Figure 4.13) with split/join annotations cannot fully replay the fitting test trace (see below the figure). The converted Petri net (Figure 4.14) is not a WF-net as it has multiple source places which is a consequent of the original tree starting with a choice. No alignments can be computed as there is no single sound execution sequence from the initial state to the end state (always remaining tokens). This results in a zero score for each trace, i.e. the fitting test traces are not completely replayable. Doing the token-based replay manually shows that it is possible to replay every activity in the trace, however, several tokens remain after reaching the end state and thus the trace is not completely replayable. This observation holds under the assumption that the log only contains complete traces. Notice the manual replay adopted a smart replay heuristic in the sense that silent transitions can only be fired if they enable a visible transition that corresponds to an activity of the test trace in the next step.

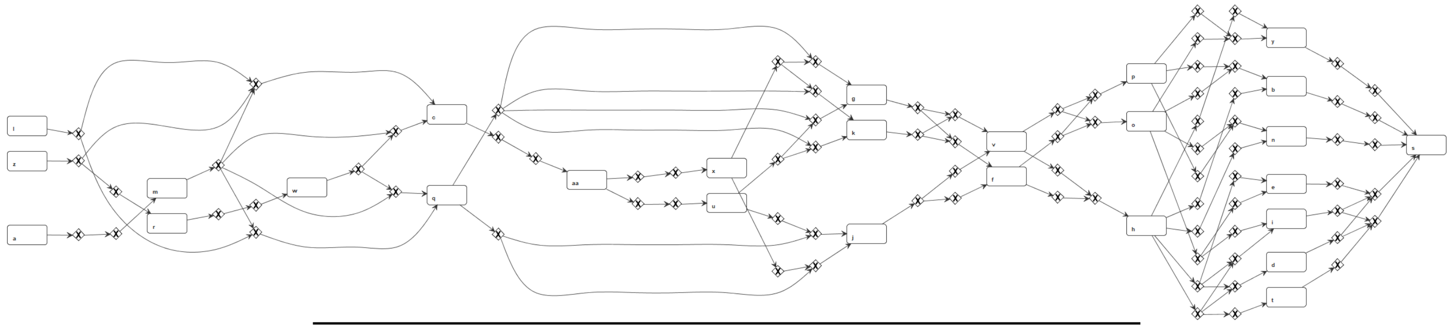
Secondly, a model was analyzed that can be aligned to the test log. The converted Petri net (Figure 4.15) is not a WF-net (because there is a loop that is not between source and sink place), but contains sound execution sequences from the initial to the final state (reason why it can be aligned). Both alignments and token-based replay classify the fitting test trace as not completely replayable because of the problem with activity "p". Therefore, if the model can be aligned, both token-based fitness and alignments result in the same trace fitness scores, i.e. completely replayable or not-completely replayable.

To conclude, the conversion from Heuristic nets to Petri nets generates (local) unsoundness which introduces problems for both alignments and token-based replay. For alignments, the problem is that in most cases there exists no single sound execution (mostly due to remaining tokens and possible deadlocks) from the initial state to the final state which makes alignments impossible. For token-based replay, the conversion with lots of invisible transitions makes the replay, with local heuristics, choose the wrong path or results in remaining tokens. A solution is to avoid the conversion to Petri nets by defining and implementing a replay technique on Heuristics nets directly (or causal nets [101] which are a new and improved representation language for Heuristics mining).

We believe that not addressing the noise, parameter and conversion aspects do not invalidate the contributions of the new procedure. While it is possible to accommodate them in the procedure, the current experiments illustrate that our procedure already properly addresses the requirements stated in the introduction of this chapter, which is further explained in the next subsection.

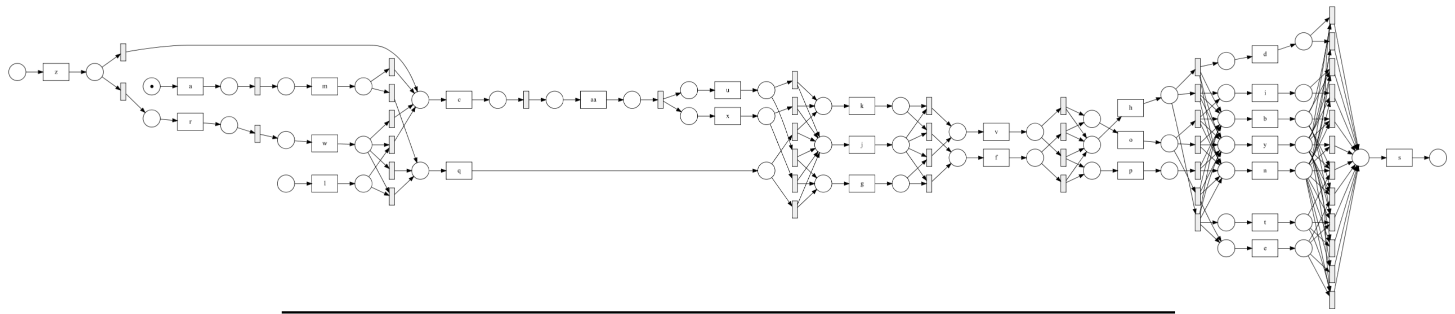
4.3.6 Requirements

The principles of Design Science research state that the evaluation should assess whether the developed artefact meets its requirements. The new procedure ful-



Test trace	a	m	c	aa	u	x	k	j	v	p	h	y	b	i	n	s
Replayable	a	m		aa	u	x	k	j	v	p	h	y	b		n	
Non-Replayable			c											i		s

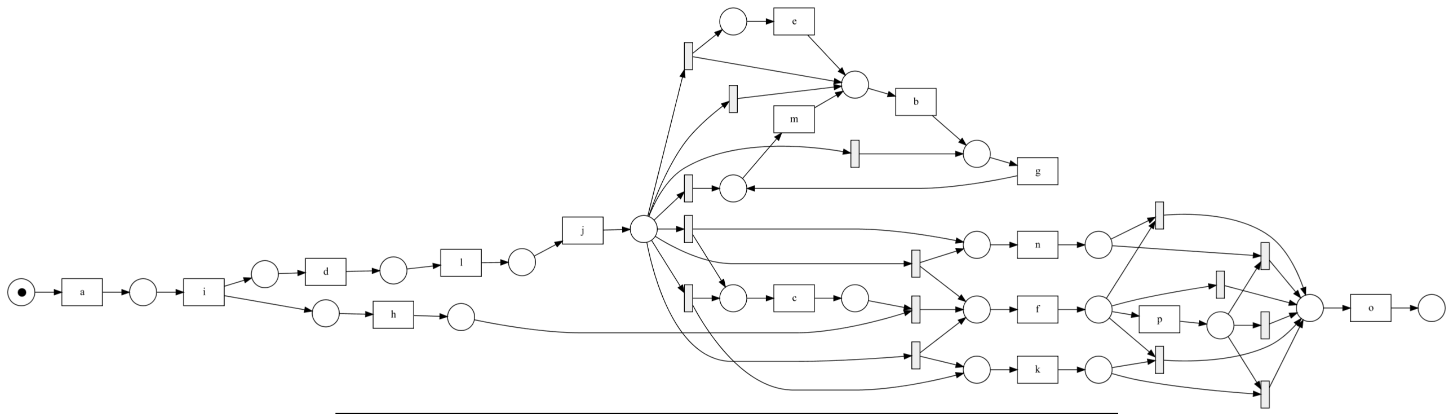
Figure 4.13: Heuristics net Discovered by the Heuristics Miner with Replay Result



Test trace	a	m	c	aa	u	x	k	j	v	p	h	y	b	i	n	s
Replayable	a	m	c	aa	u	x	k	j	v	p	h	y	b	i	n	
Non-Replayable																s*

*Trace is not completely replayable due to remaining tokens in places: pi_4, pi_7, pi_{10}, pi_{14}, pi_{25}

Figure 4.14: Converted Model without Alignments Discovered by the Heuristics Miner with Replay Result



Test trace	a	i	h	d	l	j	c	k	f	o	p
Replayable	a	i	h	d	l	j	c	k	f	o	
Non-Replayable											p*

*Alignments: P is a log move. Token-based fitness: a missing token and remaining token needed to replay P. Therefore, the trace is always classified as not completely replayable.

Figure 4.15: Converted Model with Alignments Discovered by the Heuristics Miner with Replay Result

fills the requirements stated in the introduction of this chapter: measure quality independent of modelling notation, generalize results to a model population, estimate the quality of discovery algorithm to rediscover the underlying process, and enable automated and shareable experiments.

Firstly, the new evaluation procedure is *modelling notation independent* by design. This is due to the fact that the classification approach for quality measurement only requires formal replay semantics of any used modelling notation to decide whether a given trace is completely replayable or not. As a result, the implementation requires a conformance checker for any modelling notation used by the discovery algorithms. The RapidProm implementation used for the experiments always translates the discovered models to Petri nets because the available conformance checker requires it. To illustrate, in the experiments we translated the dependency graphs discovered by the Heuristics Miner [128] and the Process trees discovered by the Inductive Miner [67] to trace equivalent Petri nets. As such the current implementation is not notation independent, however, the procedure is modelling notation independent.

Secondly, the experiments have illustrated that the new procedure allows to *specify model populations* from which random samples of models and logs can be drawn. The *size of these random samples* can be set to guarantee a certain statistical power. As a result, we can generalize the experiment findings to the defined model populations taking into account the size of the event logs, the used noise operators (see Section 4.2), and the parameter settings of the discovery algorithms.

Thirdly, there is a clear correlation between the precision and recall that we employ and the typical process-mining log measures of model quality. However, the process-mining measures are designed considering that the reference model (real model) is not known and that one only observes the positive cases, namely the traces that are part of the underlying process. The negative cases, i.e. the executions/traces that do not fit the underlying process, are not known because they would require to know the reference model. Therefore, the process-mining measures of model quality try to artificially generate the negative cases based on estimation (see, in this respect, also [118]) and, hence, the measure results may be inaccurate estimates of the discovery algorithm's quality to rediscover the underlying process. Since we know the reference process model, we can generate both positive and negative traces and label them correctly. This leads to metric results that are certain and, hence, accurate estimations of the *algorithm's quality to rediscover the underlying process*.

Finally, the scientific workflow implementation of the new evaluation procedure using RapidProM supports the *automation* of process discovery evaluation experiments. Additionally, the scientific workflows can be easily *shared* with other researchers such that they can reproduce the experiment results or extend the workflow.

4.4 Conclusion

This chapter presented a new evaluation procedure to overcome existing limitations in process discovery evaluation. The new procedure's quality measurement is independent from the discovered model's modeling notation by adopting a classification approach based on a reference model. It starts by defining a population of process models using different behavioral characteristics: workflow patterns and log characteristics. From this population a set of models and event logs is randomly sampled. Using a 10-fold cross-validation approach, the event logs are split into training and test logs. Then the procedure adds noise to half of the test traces to generate non-fitting traces. The discovery algorithm learns a model based on the training log and classifies the test traces as fitting or non-fitting. The procedure then combines the classification results in a confusion matrix together with the metrics recall, precision and F_1 score. These metrics are the input for the final statistical tests that are used to determine whether significant differences between algorithms exist or whether certain model or log characteristics have a significant effect on algorithm's quality to rediscover the underlying process.

The procedure is designed as a scientific workflow. The workflow is then implemented in the RapidMiner tool using the RapidProM extension such that the evaluation experiments can be automated, shared between researchers and extended to include new discovery techniques. The proposed procedure allows researchers to benchmark discovery algorithms as well as to perform a sensitivity analysis to evaluate whether certain model or log characteristics have a significant effect on the quality of the discovered models.

The demonstration and evaluation assessed the new evaluation procedure by conducting an extensive experiment involving four process discovery algorithms, five control-flow patterns and two levels of infrequent behavior. The experiment has shown that the new procedure supports both the benchmarking and sensitivity objectives. Furthermore, the evaluation has shown that the new procedure meets all the requirements to overcome important limitations of current evaluation approaches.

Finally, future research opportunities were identified. The first opportunity relates to adding noise to the training logs to assess its impact on algorithm's quality to rediscover the underlying process. Secondly, we could explore more noise operations to punish other imprecisions in the discovered models such as those involving non-detection of long-term dependencies. Additionally, noise insertion guided by the original model could lead to non-fitting traces that are more difficult for discovery algorithms to classify correctly. Another opportunity is to extend the binary trace-level conformance checker to a more fine-grained multi-class event-level conformance checker. Also future research could look into directly comparing the similarity between the original model and the discovered model to understand which constructs are effectively rediscovered. Furthermore, the evaluation procedure could be extended to include discovery algorithm parameter optimization as this could impact the quality of the discovered models. Finally, the RapidProM implementation should be extended with more discovery algorithms using other modelling notations such as BPMN to allow for evaluations that cover all state-of-the-art discovery techniques.

CHAPTER 5

An evaluation procedure for decision mining algorithms

During the last decade the amount of decision mining algorithms has grown. The increased number of techniques raises the need of an evaluation procedure. However, no standard empirical evaluation procedure has been developed. The lack of such a standard procedure is a result of the following remaining challenges related to the evaluation of decision mining techniques (identified in Section 2.2.3):

1. There is a lack of a data set selection method. Only a few real-life and manually created artificial event logs have been used to evaluate decision mining techniques which has limited the generalizability of the evaluation results. Firstly, the population from which these real-life event logs come is unknown, and therefore, we do not know the type of process behavior these logs contain. Routing decisions that depend on case attributes may be fully-deterministic or non-deterministic. The influence of such characteristics must be acknowledged as they may affect the quality of the decision miner's output. Secondly, the manual creation of artificial models and logs resulted in non-random inclusion of process characteristics in

the generated processes. The non-randomness, together with the limited number of logs, impedes the statistical generalizability of the evaluation results.

2. The quality measurement of the decision miner's ability to rediscover the routing decision rules of the underlying process is typically quantified by comparing the event log with the discovered routing decision rules. Such log measures make assumptions on the completeness and the noise of/in the log and therefore may bias the calculated quality score.

This chapter introduces an empirical evaluation procedure for decision mining to overcome the above described challenges by:

1. Incorporating the GED methodology and "DataExtend" implementation (introduced in Chapter 3) as a method for data set selection tailored towards the empirical evaluation of decision mining techniques. The GED methodology starts from a population of processes with varying behavioral characteristics from which a random sample of process models and event logs with routing decision rules based on case attributes can be drawn. Such a random sample enables the generalization of the decision mining evaluation results to the known process population;

2. Using the knowledge of the reference model to get an unbiased estimate of the decision miner's quality to rediscover the routing decision logic of the underlying process.

The proposed procedure is an extension of the process discovery evaluation procedure presented in Chapter 4 and it meets the requirements defined in Section 2.3.3. The material in this chapter is based on the work published in [57]. This chapter starts by describing the extension to the discovery evaluation procedure to allow for decision mining evaluation. This is followed by an experiment of two decision mining techniques to demonstrate and evaluate the new procedure. Finally, the conclusion sums up the contributions and future research opportunities.

5.1 A decision mining evaluation procedure

The proposed procedure focuses on evaluating the quality of decision mining algorithms to rediscover the routing decision logic of the underlying process when given a part of all its possible behavior. The procedure is an extension of the evaluation procedure for process discovery algorithms as presented in Chapter 4. In the discovery procedure the focus was on the control-flow perspective of the process, while the extension combines the data and control-flow perspectives. More specifically, the routing decisions of the process can depend on case attribute values. To incorporate these changes, several changes to the design and building blocks of the original workflow for discovery evaluation are needed and discussed in more detail below.

5.1.1 Design of the decision mining procedure

Similar to the discovery procedure, the decision mining procedure enables the following evaluation objectives: benchmarking different decision mining algorithms, and performing a sensitivity analysis to study the effect of model and log characteristics on algorithm performance. Figure 5.1 shows the decision mining evaluation procedure as a workflow. The decision mining workflow starts from a predefined evaluation objective and then carries out the data generation, decision mining, quality measurement and statistical analysis steps. The decision mining procedure uses a classification approach as this benefits the use of reference model knowledge in the quality measurement step which is one of the two requirements stated in the introduction of this chapter.

In a first step, based on the evaluation objective, the researcher determines the model population that is used during the data generation step. The model population specifies the control-flow *and* data-flow relating to the routing decisions of the generated processes.

Then, for each decision mining technique we draw a random sample of control-flow models from the population. Then, for each "original model" in the random sample, the decision mining procedure adds random case attribute dependencies ("rules") to the routing decisions. Subsequently, the model enhanced with these "rules" is simulated in an event log, i.e. a random sample of cases

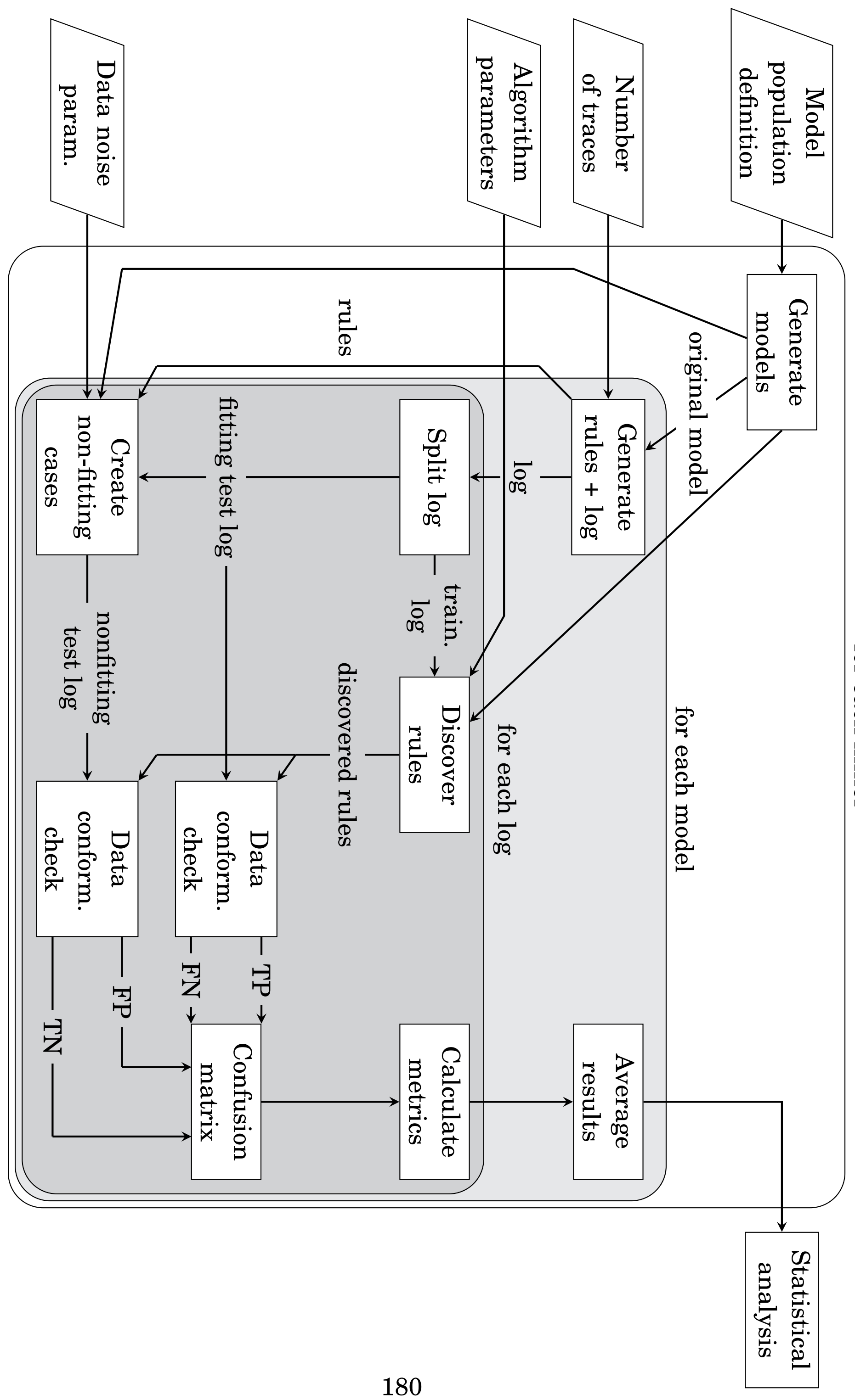


Figure 5.1: Overview of the decision mining evaluation procedure

from all possible cases allowed by the model and "rules". The "rules" on top of the routing decisions of the "original model" establish the "ground truth". The procedure assesses decision mining techniques that aim to rediscover the "rules" given a control-flow model and an event log with case attributes. To test the rediscovery of "rules", the procedure provides the "original model" and an event log containing all the attributes contained in the "rules". Therefore, the evaluation procedure does not test the decision mining techniques' robustness to deal with noise or incompleteness, i.e. incorrect control-flow models or with event logs with incorrect/incomplete case information. Such robustness tests are a possible future extension to the proposed evaluation procedure.

The following steps apply the 10-fold cross validation approach for obtaining precise estimates of the decision miner's quality to rediscover the "rules". Each generated log is split ("split log") into 10 folds of equal sizes, where 9 folds together form the training log and the other fold constitutes the test log. The training log together with the "original model", that contains only the control-flow behavior, are the inputs of the decision mining algorithm ("discover rules"). The algorithm aims to discover the routing decisions' dependencies based on case attributes in the training log. In a next step, the procedure uses a classification approach to evaluate the discovered rules. This requires a test log with both positive and negative examples.

Up till now, the test log only contains positive examples. Therefore, half of the cases in the test log are modified by task "create non-fitting cases" to make them non-replayable by the "original model" with "rules". The cases are non-fitting with regard to the data-flow only and not the control-flow as the procedure evaluates the rediscovery of "rules" as indicated above. Then, the "data conformance checking" tasks replay both the fitting and non-fitting test cases on top of the "original model" with the "discovered rules" to classify the test cases. The classification results are combined in a confusion matrix which is used to calculate the recall, precision and F_1 scores. These metrics quantify the quality of the decision mining technique. More specifically, recall measures how much fitting behavior in the test log is classified as fitting by the "original model" with "discovered rules", and precision measures how much behavior classified as fitting by the "original model" with "discovered rules" is actual fitting behavior in the test log. Decision mining techniques aim to discover rules that ensure good recall such that they allow for the behavior in the underlying process, and at the same time guarantee good precision, i.e. balance between overfitting and underfitting the behavior in the training log. The F_1 score combines the recall and precision metrics into one value that simplifies the comparison of evaluation results.

After repeating the decision mining and quality measurement steps for the 10 folds, the “average results” task takes the mean of the recall, precision and F_1 scores over the 10 folds. As such the decision mining procedure aims to get an unbiased estimate of the true quality of a decision mining technique to rediscover the “rules”. Finally, the statistical analysis uses the evaluation results to test the hypotheses stated in the evaluation objective(s).

The design of this decision mining procedure, similar to the discovery procedure (see Section 4.1.1), features that no two algorithms are applied on the same event log. Moreover, each log is drawn from a different combination of “rules” and process model which are randomly drawn from a model population. Therefore, the “discovered rules” and quality metrics are independent observations which enables a less complex statistical analysis than when observations are dependent. Furthermore, the independence setup fits the procedure’s focus on evaluation rather than prediction of the best algorithm given some model or log characteristics. The latter typically requires dependent observations.

The above setup of evaluating the decision mining algorithms using a classification approach seems quite complex given the alternative of directly comparing the original “rules” with the “discovered rules”. The following example illustrates that the alternative approach also presents a challenge. Consider the original model in Figure 5.2 where the choice between activities “d” and “h” is denoted as routing decision 1 and the choice between activities “c”, “b”, “e” and “f” is denoted as routing decision 2. The target determinism level of the model with case attribute dependencies (“rules”) is set to 0.5.

The original “rules” for routing decision 1 are as shown in Table 5.1, and the original “rules” for routing decision 2 are as shown in Table 5.2.

The original model with “rules” is then simulated into an event log with 1000 cases. The resulting event log and original model are used as input of the

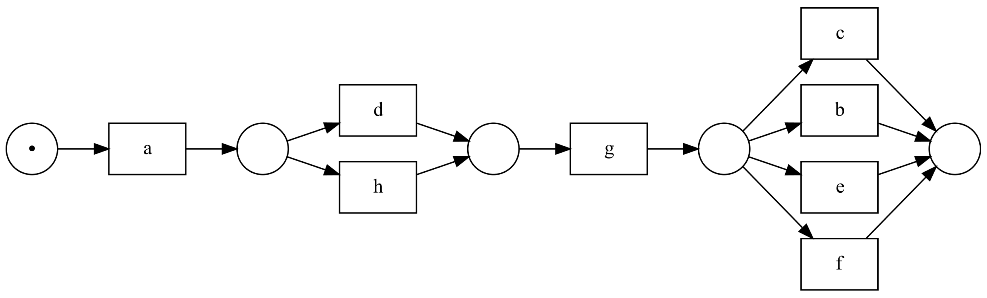


Figure 5.2: An example original process model.

	inputs			*output*
Rule number	**Z_1**	**Z_2**	**Z_3**	**Routing decision 1**
1	true	true	true	h
2	true	true	false	d
3	true	true	false	h
4	true	false	true	h
5	true	false	false	d
6	true	false	false	h
7	false	true	true	d
8	false	true	true	h
9	false	true	false	h
10	false	false	true	d
11	false	false	true	h
12	false	false	false	d
13	false	false	false	h

Table 5.1: Decision table for first routing decision

	input	*output*
Rule number	**Routing decision 1**	**Routing decision 2**
1	d	c
2	d	b
3	h	c
4	h	e
5	h	f

Table 5.2: Decision table for second routing decision

				inputs	*output*
Rule number	**Z_1**	**Z_2**	**Z_3**	**Routing dec 1**	**Routing dec 2**
1	-	-	-	d	b
2	-	-	-	d	c
3	false	-	false	h	c
4	true	false	false	h	c
5	-	-	false	h	e
6	-	true	true	h	e
7	true	true	false	h	f
8	-	false	true	h	f

Table 5.3: Discovered decision table for second routing decision

overlapping rules decision mining technique [74]. The overlapping technique cannot discover rules for routing decision 1, instead it uses all available case attributes as inputs of routing decision 2, shown in Table 5.3.

The discovered rules of routing decision 2 in Table 5.3 cannot be compared directly to the original rules related to routing decision 2 in Table 5.2 as it has a different set of input case attributes: routing decision 1, Z_1, Z_2, Z_3 versus only routing decision 1. To overcome this issue, one has to combine the rules in Table 5.1 and Table 5.2 to compare them with the discovered rules in Table 5.3. The results are combined per routing decision output:

- Discovered rule 1 in Table 5.3 is the same as the original rule 2 in Table 5.2: if routing decision 1 executed activity “d”, then activity “b” should be chosen in routing decision 2.

- Discovered rule 2 in Table 5.3 corresponds to the original rule 1 in Table 5.2, i.e. if routing decision 1 executed activity “d”, then activity “c” should be chosen in routing decision 2. Discovered rules 3 and 4 of Table 5.3 are not similar to original rule 3 in Table 5.2. This conclusion relies on checking if discovered rules 3 and 4 cover all the original rules in Table 5.1 that have activity “h” as output, i.e. rules 1, 3, 4, 6, 8, 9, 11 and 13. Original rules 1, 3, 4, 8 and 9 are not covered, thus the decision miner cannot completely rediscover the original rule 3 in Table 5.2.

- Similar to the above observation, the discovered rules 5 and 6 in Table 5.3 do not cover the original rule 4 in Table 5.2 by checking all the original rules in Table 5.1 that have activity “h” as output.
- Similar to the above observation, the discovered rules 7 and 8 in Table 5.3 do not cover the original rule 5 in Table 5.2 by checking all the original rules in Table 5.1 that have activity “h” as output.

To conclude, directly comparing the discovered rules with the original rules often requires a complex combination of rules in different tables. Therefore, the current setup of replay that is an extension of the discovery evaluation procedure of Chapter 4 is chosen instead.

The following subsection describes the procedure tasks that differ from the discovery evaluation procedure.

5.1.2 Adapted building blocks of the decision mining procedure

This section focuses on building blocks that differ from the blocks described in Section 4.1.2 related to the discovery procedure.

5.1.2.1 Generate rules and log

For each random control-flow model drawn in the “generate models” block, this block will first enhance the routing decisions in the model randomly with case attribute information to make routing decision dependencies (“rules”). In the second step the “rules” with the model together are simulated in an event log that contains a random set of all possible cases allowed by the model with “rules”. The user influences the enhancement step by specifying the target determinism (part of the model population, see Section 3.4.2.1) of the routing decisions that depend on case attribute values. As such one can control whether routing decisions are fully-deterministic, i.e. when target determinism is equal to 1, or not when $0 <$ target determinism < 1. A fully deterministic routing decision allows for exactly one output branch given the case attribute values. A non-deterministic decision, on the other hand, allows for one or more output

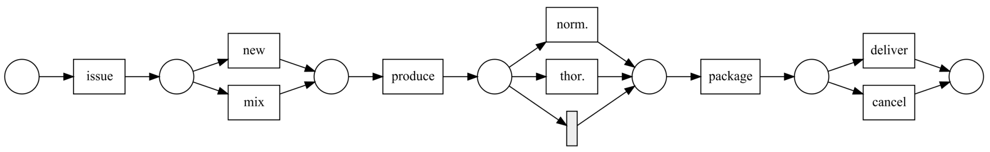


Figure 5.3: Petri net representing make-to-order example process. Activity names are abbreviated: “issue”: issue order, “new”: prepare new materials, “mix”: prepare mixed materials, “produce”: produce order, “norm.”: inspect normally, “thor.”: inspect thoroughly, “package”: package products, “deliver”: deliver products, “cancel”: cancel delivery.

branches given the case attribute values. Additionally, the user specifies the number of cases to generate from the model enhanced with “rules”.

5.1.2.2 Create non-fitting cases

The classification approach used for evaluating the quality of decision mining algorithms requires test logs with positive and negative examples. The test log that was created by the “split log” task only contains cases that fit the “original model” with “rules” (positive examples). The “create non-fitting cases” modifies half of the cases in the test log to create non-fitting cases (negative examples). With these non-fitting cases the procedure aims to punish “discovered rules” that are overly general, i.e. they allow to execute too many outgoing branches of a routing decision. An extreme example of this is when the “discovered rules” allow to execute any outgoing branch of a routing decision which in reality is restricted by case attribute values. To illustrate this, consider the make-to-order process in Figure 5.3 together with the “rules” for the last routing decision involving the choice between activities “deliver” and “cancel” in Table 5.4. If a decision miner is given the two cases in Table 5.5 and cannot discover the original “rules” of the last routing decision, i.e. for any case attribute value the either activity “deliver” or “cancel” can be executed, then the decision miner overgeneralizes the behavior in the log.

Given the “original model”, “rules”, and a test case, the case attributes of that case are changed until the case is not replayable by the “original model” and “rules”. Once the case is non-fitting, it is not modified anymore. For example,

Rule number	*input* **Acceptable Quality?**	*output* **Routing decision 3**
1	true	deliver
2	false	cancel

Table 5.4: Decision table for third routing decision

Case ID	Event ID	Activity	Routing Decision 1	Acceptable Quality?	Premium?
1	1	issue			true
1	2	prepare mix	prepare mix		true
1	3	produce	prepare mix		true
1	4	inspect thoroughly	prepare mix	true	true
1	5	package	prepare mix	true	true
1	6	deliver	prepare mix	true	true
2	7	issue			false
2	8	prepare new	prepare new		false
2	9	produce	prepare new		false
2	10	package	prepare new	false	false
2	11	cancel	prepare new	false	false

Table 5.5: Example cases of the make-to-order process.

if the value of case attribute “acceptable quality” is changed to “true” for the second case, then it no longer fits the rules in Table 5.4, and therefore, it becomes a non-fitting case. In that way, the noise generation punishes the overgeneral decision mining output that allows for activity “cancel” no matter what the case attribute values are.

Notice that this task does not modify the trace of a case as done by the “create non-fitting traces” of the discovery procedure (see Section 4.1.2.4). Surely the aim of the decision mining procedure lies on evaluating the “discovered rules” given the routing decisions in the “original model”. Modifying the trace of a case by swapping, duplicating or deleting activities would always be classified correctly as non-fitting by the given “original model” and would not provide any information on the quality of the “discovered rules”.

5.1.2.3 Discover rules

This block applies a decision mining algorithm on the original control-flow model together with the “training log”. The output of the decision miner is a model enhanced with “discovered rules” that explain the routing decision logic. A user can specify the parameter settings of each algorithm employed.

5.1.2.4 Data conformance checking

The data conformance checker will replay the cases of the fitting and non-fitting test logs on the “original model” with the “discovered rules”. It will consider both the control-flow and data perspectives of the case in contrast to the conformance checker used in the discovery procedure in Section 4.1.2.6. Notice that, although the non-fitting cases are only with regard to the data perspective, the conformance checker requires both perspectives to actually replay the case on the model enhanced with rules. The data-aware replay will classify a case as fitting if it can be completely replayed by the “original model” with the “discovered rules”, otherwise a case is classified as non-fitting. Subsequently, identical to the discovery procedure, the decision mining procedure combines the classification results in a confusion matrix to compute the recall, precision, and F_1 metrics.

The next section will discuss a demonstration and evaluation of the proposed procedure to evaluate decision mining algorithms.

5.2 Demonstration and evaluation

The demonstration validates the proposed evaluation procedure through an experiment that empirically analyzes two decision mining algorithms: the approach introduced by de Leoni et al. [28] that discovers mutually-exclusive (fully-deterministic) routing decision rules based on case attributes in the input event log, and an extension to that approach introduced by Mannhardt et al. [74] that allows to discover overlapping routing decision rules (i.e. non-deterministic routing decisions). The goal of the new decision mining evaluation procedure is to analyze and compare the quality of decision mining algorithms to rediscover the routing decision rules of the underlying process based on observed executions (event logs). The routing decision rules to be rediscovered can be part of processes that come from different populations by varying the probability of occurrences of control-flow process characteristics such as exclusive choices, loops and infrequent paths, and, additionally, by varying data-flow process characteristics, e.g. the determinism of routing decisions influenced by case attributes. The experiment tests decision mining algorithms on event logs coming from different model populations by varying the determinism of routing decisions and enabling or disabling the presence of infrequent paths. In this way, we can study the impact of infrequent behavior and of different determinism levels on the quality of decision mining techniques.

The evaluation argues why the decision mining procedure alleviates the remaining challenges for decision mining evaluation. It does this by discussing how the proposed evaluation procedure fulfills the requirements in the introduction of this chapter: enable generalization of results to the model population and estimate the miner's ability to rediscover the routing decision logic of the underlying process. As such, the goal of the evaluation experiment is to validate the proposed procedure and does not serve as a benchmark of all current state-of-the-art decision miners.

The remainder of this section first describes the setup of the experiment followed by a statistical analysis of the experiment results. Then, a discussion explains the experiment results, its implications, and possible future improvements to the decision mining procedure. Subsequently, limitations of the executed experiment and the threats to validity of the results are discussed. Finally, the section ends with an evaluation of the requirements that the proposed procedure aims to fulfill.

5.2.1 Experiment setup

To automate the experiment, the decision mining evaluation procedure was operationalized in the ProM framework [120]. The "DataExtend" method (see Section 3.4) is used for generating random process trees from a model population and extending the exclusive choice operators with case attributes which are then simulated into event logs. The tested decision mining algorithms, from now on referred to as *mutually-exclusive rules* [28] and *overlapping rules* [74], are available in ProM and return Data Petri nets (see Section 2.1.2).

5.2.1.1 Applied decision mining algorithms and conformance checking

The *mutually-exclusive rules* technique [28] requires a control-flow model (Petri net) and an event log as inputs to learn the decision logic of the routing decisions in the model. A crucial assumption is that the event log contains case attributes which influence the routing decisions. The decision mining technique first aligns the event log with the control-flow model such that the trace of every case in the log has a corresponding path in the model. As such the alignments mitigate the effects of non-conforming process models and also invisible activities in the model, i.e. activities that have no corresponding event in the log, but are often routing decision outcomes (skipping activities). Then, the technique aims to rediscover the decision rules that explain the routing decisions in the model using the case attributes in the log. The rule discovery is turned into a classification problem: the routing decision outcomes are the target classes and the case attribute values before the routing decision was reached (based on alignments) are used as features. The decision mining technique learns a decision tree using the C4.5 algorithm [85]. In those decision trees the leaves are the routing decision outcomes such that there can be multiple leaves for one outcome. Finally, the decision mining technique builds a decision rule for each leaf by taking the conjunction (and) of the conditions from the corresponding leaf nodes to the root node in the learned decision tree. If more than one leaf corresponds to one routing decision outcome, the found rules are combined in disjunction (or). This results in mutually-exclusive decision rules for each routing decision outcome.

The *overlapping rules* technique [74] extends the *mutually-exclusive rules* technique to discover overlapping decision rules, i.e. non-deterministic routing decisions. The extension deliberately trades the precision of the discovered rules for fitness such that less cases in the given event log are misclassified, i.e.

	inputs		*output*
Rule	**Costs**	**Check**	**Routing decision**
1	< 500	-	a
2	≥ 500	-	b
3	≥ 500	true	a

Table 5.6: Example of overlapping rules discovery. The hyphen indicates that the rule is indifferent with regards to the value of that case attribute.

violate the discovered rules. The *overlapping rules* technique starts by building a decision tree for each routing decision similar to the *mutually-exclusive rules* technique. Then, it extracts the cases in which a routing decision is wrongly classified by the learned decision tree. For the wrongly classified instances, it learns a new decision tree that yields new rules that are used in disjunction (or) of the initial rules to get overlapping rules. Consider a routing decision with two outcomes "a" and "b" with attributes "costs" and "check" that influence it (see Table 5.6): if costs are lower than 500, "a" happens, if costs are higher than 500, a possible overlap between "a" and "b" exists depending on whether the check equals "true". The *mutually-exclusive rules* technique can only rediscover the first two rules, while the *overlapping rules* technique allows to rediscover the last rule.

In the experiment, both decision mining algorithms are applied with their standard configurations. To do conformance checking on Data Petri nets, we use the multi-perspective alignment approach of de Leoni et al. [27].[1]

5.2.1.2 Artificial data generation setup

As mentioned above, the experiment analyzes the impact of infrequent behavior and of different determinism levels on the quality of decision mining techniques

[1]We used the non-balanced alignment approach [27] over the balanced alignment approach [73] to compute the alignments as it is much faster in computation time. The non-balanced approach computes the alignment in two stages: first the control-flow and then the data-flow. This does not guarantee a balance between the two perspectives when deviations can be explained in both the control-flow and data-flow perspectives. However, in our experiments, the only deviations are with regard to the data-flow and therefore it is safe to use the non-balanced approach.

Decision mining Technique	**Infrequent Paths**	**Determinism Level**
Mutually-exclusive rules [28]	False	0.5, 0.75
Overlapping rules [74]	True	1

Table 5.7: Summary of the possible values of the three independent variables included in the experimental setup: 12 ($2 \times 2 \times 3$) value combinations. The determinism level indicates whether choices in the process tree that depend on case attributes are fully-deterministic (value of 1) or non-deterministic (value below 1).

to rediscover the routing decision logic of the underlying process. Therefore, the experimental design includes all the combinations of three independent variables: decision mining technique used, presence or absence of infrequent paths and the level of determinism. The three variables and their levels are summarized in Table 5.7. In total, the 12 possible combinations are included in the experiment: 2 decision mining techniques $\times$ 2 levels of infrequent behavior $\times$ 3 levels of determinism.

The presence/absence of infrequent paths and the determinism level are varied as these are part of the independent variables. The other process characteristics are fixed for each model population. The probability of the sequence, exclusive choice and parallelism patterns is fixed at values 46%, 35% and 19%, respectively. These probabilities are based on the analysis of a large collection of models as reported by Kunze et al. [65]. "DataExtend" only introduces routing decision rules based on case attributes to exclusive choice operators in the tree. Therefore, we leave out operators that lead to routing decisions for which no rules are introduced, i.e. the probabilities of "loop" and "or" patterns are set to zero. The size of the models within each model population varies between 6 and 10 activities, with a mode of 8 activities. This makes the six model population definitions as follows: MP_{data} as shown in Table 5.8 where X and Y are assigned all 6 combinations of values in column two and three in Table 5.7.[2]

Furthermore, we have specified the case attributes introduced by "DataEx-

[2] $\Pi^{In} = 0$ for "False" (absence of infrequent paths) and $\Pi^{In} = 1$ for "True" (presence of infrequent paths).

Parameter	**Population** $\mathbf{MP_{data}}$
Number of visible activities	(6,8,10)
Sequence ($\Pi^{\rightarrow}$)	0.46
Parallel ($\Pi^{\wedge}$)	0.19
Choice ($\Pi^{\times}$)	0.35
Loop ($\Pi^{\circlearrowleft}$)	0
Or ($\Pi^{\vee}$)	0
Silent activities (Π^{τ})	0
Reoccurring activities (Π^{Re})	0
Long-term dependencies (Π^{Lt})	0
Unfold loops	n/a
Max repeat (k)	n/a
Infrequent paths (Π^{In})	Y
Sample size (number of trees)	129
Logs per model	1
Number of traces (t)	[200,1000]
Noise (Π^{Noise})	0
Determinism level	X
attribute type	$\in$ {boolean, string, numerical}
# intervals	$\sim uniform(1,4)$
# assigned attributes	$\sim uniform(0,3)$

Table 5.8: Model population parameters for the experiments, where X and Y are assigned all 6 combinations of values in {0,5.75,1} and {0 (False),1 (True)} respectively.

tend" in the following way. Firstly, the introduced case attributes are of three different types: boolean, string and numerical. Each numerical attribute is discretized to intervals to make a finite number of decision rules at each routing decision. The number of intervals i is randomly drawn from a discrete uniform distribution: $\#i \sim \text{uniform}(1,4)$. For example, consider a numerical variable V_1 with values between 0 and 1 that has two intervals, i.e. one random cutoff point $x = 0.6829$ is chosen which results in the intervals $[0, 0.6829)$ and $[0.6829, 1)$. Secondly, we limit the number of case attributes that influence a routing decision to three. This means that the number of assigned case attributes $\#c$ of a routing decision follows a discrete uniform distribution: $\#c \sim \text{uniform}(0,3)$. This means that each routing decision is assigned zero or more case attributes (see also Definition 3.6 in Section 3.4.2.1). These settings for the case attributes further refine the model population that is used during the experiments and as such to which population we can generalize the results. Notice that the implementation of "DataExtend" also allows for other values for the uniform distributions specified for $\#i$ and $\#c$.

As specified by the evaluation procedure's design (see Section 5.1.1), we draw a random sample of 129 control-flow models for each miner from each of the six model populations. Then, "DataExtend" randomly extends each model with routing decision rules. Each model with rules is simulated into an event log containing between 200 and 1000 cases. As a result, each quality measurement is an independent observation that allows for a fixed effects ANOVA analysis [122] to study the effect of decision mining technique, infrequent paths, and different levels of determinism. The sample size of 129 models results in a statistical power of $1-\beta = 0.95$ when using a significance level α equal to 0.05.[3] The power indicates that there is a 95% probability to detect a difference between decision mining techniques when actually a relatively small difference exists. In total, 1548 models, sets of routing decision rules, and logs are generated: 6 populations $\times$ 2 decision miners $\times$ 129 models = 1548 models (with one set of routing decision rules and one log per model).

Finally, we have calculated the completeness of the generated logs with

[3]The power was computed with the G*Power tool [38].

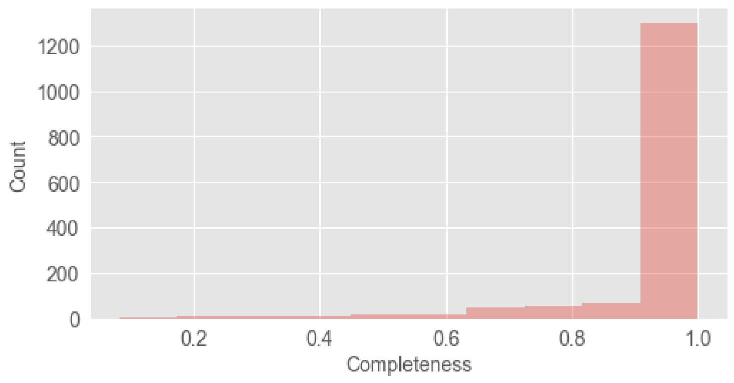


Figure 5.4: Distribution of completeness of logs wrt. their respective process models. Completeness is measured as the fraction of traces allowed by the model that are present in the event log.

regard to all possible paths in the model. The completeness is the proportion of unique traces in the log to all possible unique traces according to the model using the technique described in [49]. Fig. 5.4 shows that the majority of the logs is complete with regard to the behavior in the model. This was caused by the relatively high number of cases, i.e. between 200 and 1000, compared to the size of the models in terms of activities: minimum 6 activities, maximum 10 activities and a mode of 8 activities.

5.2.2 Analysis of the results

To study the effects of decision mining techniques, infrequent paths and different levels of determinism we use a one-way ANOVA analysis *if* the assumptions of homogeneity of variances and normality of the dependent variable hold [122]. However, at least one of the assumptions is violated for every dependent variable, i.e. F_1, recall and precision. Therefore, the non-parametric *Kruskall-Wallis* test (KW), multiple comparison post hoc test, and Jonckheere test [94] are applied instead (see Section 4.3.2.1 for a description of these tests).

	Mutually-exclusive rules	**Overlapping rules**
Recall	796.34	752.66
Precision	776.63	772.37
F_1 score	789.52	759.48

Table 5.9: Average Ranks per Miner. Each cell indicates the average ranking for a specific quality dimension (row header) and for a specific miner (column header). One can compare miners by comparing the average ranks within one row.

5.2.2.1 The effect of decision mining technique

We aim to learn the effect of a decision mining technique on each of the dependent variables: recall, precision and F_1 score. The effects of the other independent variables, i.e. infrequent paths level and determinism, are not studied here.

We apply the KW method, to test whether the average rank differs between the two decision mining techniques (i.e. samples). This required us to rank all the 1548 averages taken over the 10-fold cross validation for recall, precision and F_1 values ignoring sample membership (i.e. decision mining technique). The highest value for recall, precision and F_1 gets rank 1 (lowest rank), while the lowest absolute value gets rank 1548 (highest rank). Then we computed the average ranking per miner, i.e. the average position of a discovered model with rules by that miner for that quality metric on a scale from 1 to 1548. A higher average ranking means worse performance. The ranking summary is shown in Table 5.9.

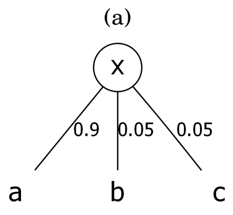
The rankings show relatively small differences in average rankings between the two miners in all quality dimensions. The order suggested in all dimensions is: *overlapping* > *mutually-exclusive*, which means that the *overlapping* technique discovers the best routing decision rules in terms of recall, precision, and F_1 scores. Based on the KW test, only the difference between miners in terms of recall is statistically significant at a 5% significance level. See Table B.7 in Appendix B for a summary of the statistical test results for the recall, precision, and F_1 scores.

5.2.2.2 The effect of infrequent paths

The analysis tests whether the presence/absence of infrequent paths has an impact on the average ranking of the two decision mining techniques for recall, precision and F_1 scores. The effect of determinism level is not studied here.

Before the actual analysis, we want to make an important caveat on the effect of the infrequent paths parameter relating to the interplay between the control-flow and data-flow perspective during the model, rules, and log generation by "DataExtend". Infrequent paths are denoted with an imbalance in execution probabilities of the output-branches of each exclusive choice construct in the model. The effect of these execution probabilities depends on whether the exclusive choices have routing decision rules based on case attributes attached. In case there are no rules attached, the imbalance of the execution probabilities always results in infrequent traces in the log. In the other case, when exclusive choices have routing decision rules, the execution probabilities only matter when the routing decisions are non-deterministic. Non-deterministic routing decisions allow for more than one outgoing branch based on the case attribute values. In that case, the execution probabilities are taken into account when deciding which branch is activated. Nevertheless, the case attribute values are drawn from a (discrete) uniform distribution. As a result, the imbalance between traces might be mitigated. As an example consider the process tree and routing decision rules in Figure 5.5. There exists an imbalance in execution probabilities with a 90% probability to execute activity "a". However, on average only 45% of the process executions will contain "a" as there is a 50% probability that case attribute "X" equals "true". We acknowledge that stepping away from the uniform distributions could introduce proper infrequent paths. Yet, this presents a new challenge when case attributes are influencing more than one routing decision. The current experiment uses uniform distributions for case attribute values, but future experiments should consider other distributions as well to test infrequent paths.

In the analysis, we first split the sample into two subsets: experiments with infrequent behavior and experiments without infrequent behavior. This division is called *blocking* (see Section 2.3.2.2) which is done to isolate the variation in



(b)

Rule	*input* X	*output* Routing decision
1	true	a
2	true	b
3	false	c

Figure 5.5: Infrequent paths with case attribute dependencies: (a) shows the process tree with infrequent paths, (b) shows the decision table with routing decision rules.

	Mutually-exclusive rules	**Overlapping rules**
Recall	395.80	379.20
Precision	395.42	379.58
F_1 score	400.14	374.86

Table 5.10: Average ranks per miner with infrequent behavior

	Mutually-exclusive rules	**Overlapping rules**
Recall	400.99	374.01
Precision	381.63	393.37
F_1 score	389.58	385.43

Table 5.11: Average ranks per miner without infrequent behavior

recall, precision and F_1 scores attributable to the absence/presence of infrequent paths. Secondly, the KW test is applied to each subset.

Tables 5.10 and 5.11 contain the average rankings per decision mining technique grouped by metric and experiments with and without infrequent behavior respectively. The average rankings denote the average position of a discovered model with rules by that miner for that quality metric on a scale from 1 to 774. A higher average ranking means worse performance. These rankings indicate that in all cases the *overlapping* technique outperforms the *mutually-exclusive* technique except for precision when there is no infrequent behavior.

Based on the KW test we can conclude that none of the differences in rankings between the two miners is statistically significant (see Tables B.8 and B.9 in Appendix B). As such, there is no statistical evidence that the presence/absence of infrequent paths influences the two decision mining techniques. Notice that this observation holds given that infrequent paths are not guaranteed in all situations as described above.

5.2.2.3 The effect of determinism level

The analysis investigates how the quality of each decision mining technique (in terms of precision, recall and F_1 score) is influenced by the determinism level of routing decisions: a value of 1 results in fully-determistic routing decisions, while values below 1 result in non-deterministic routing decisions. The effect of infrequent behavior is not studied here.

Figures 5.6, 5.7, and 5.8 illustrate the average recall, precision, and F_1 scores for all the decision mining techniques over different determinism levels. The bars indicate the 95% confidence interval for the averages. The graphs indicate a (small) positive trend, i.e. increasing the determinism level has a positive effect on recall, precision, and F_1 scores, except for the recall scores of the *overlapping* technique that remain relatively steady over the different levels. To determine whether the positive trend is statistically significant, an in-depth analysis is performed.

First, the sample is divided into subsets grouped by decision mining technique. As such, the variation in accuracy associated with the decision mining technique is isolated. Then, similar to the analysis above, the KW test is applied to compare the average rankings of the discovered routing decision rules.

Table 5.12 contains the average ranks for the *mutually-exclusive* technique for all three metrics per determinism level. The average rankings seem to suggest that as the determinism level increases, the recall, precision, and F_1 scores also increase. To test this impression statistically, we have relied on the KW and Jonckheere tests. Both tests confirm there is a statistically significant positive trend in the recall and F_1 quality dimensions. A pairwise comparison shows that the differences between fully-deterministic (determinism of 1) and

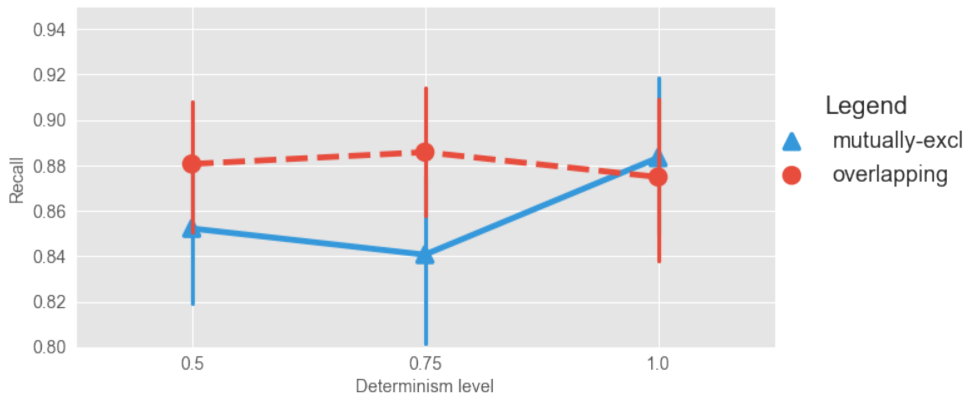


Figure 5.6: Recall scores for decision mining techniques for different levels of determinism.

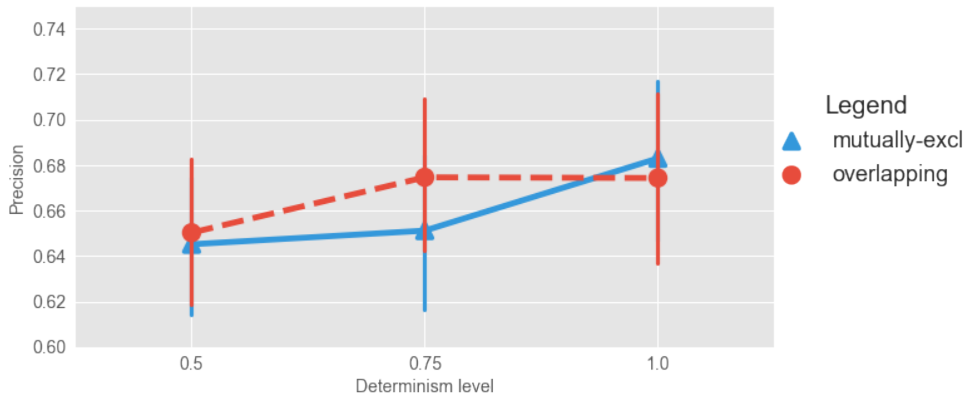


Figure 5.7: Precision scores for decision mining techniques for different levels of determinism.

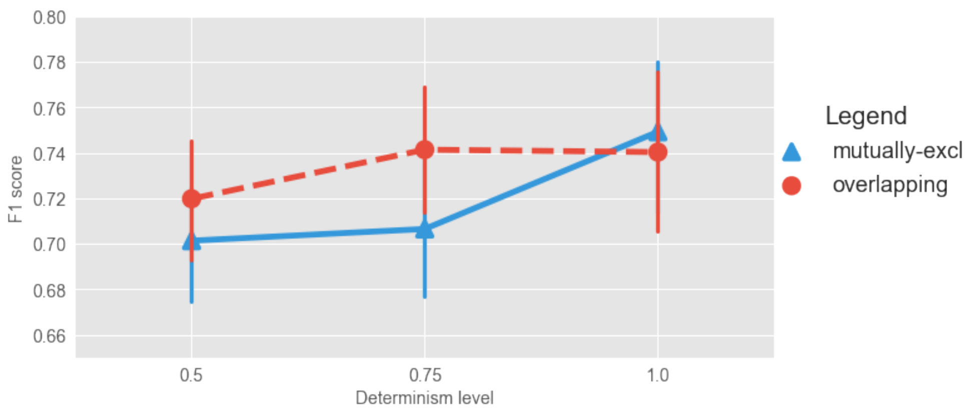


Figure 5.8: F_1 scores for decision mining techniques for different levels of determinism.

Determinism level	**0.5**	**0.75**	**1**
Recall	407.11	406.83	348.56
Precision	407.41	391.95	363.13
F_1 score	421.37	400.67	340.46

Table 5.12: Average ranks of mutually-exclusive technique per determinism level.

Determinism level	**0.5**	**0.75**	**1**
Recall	406.72	393.19	362.59
Precision	409.93	378.85	373.73
F_1 score	422.43	381.08	358.99

Table 5.13: Average ranks of overlapping technique per determinism level.

non-deterministic routing decisions (determinism of 0.5 or 0.75) are statistically significant (see Table B.10 in Appendix B). For the precision dimension, the differences are not statistically significant.

Table 5.13 contains the average ranks for the *overlapping* technique for all three metrics per determinism level. The data suggests relatively small positive trends in both recall and precision dimensions, and a somewhat larger positive trend in terms of F_1 score. The KW and Jonckheere test reveal no statistically significant positive trend for recall and precision. The positive trend for F_1 score is statistically significant. However, the pairwise comparisons of determinism levels only confirms this trend between the largest difference in determinism levels, i.e. between 0.5 and 1 (see Table B.11 in Appendix B).

5.2.3 Discussion

The graphs and the analysis of the effect of decision mining technique highlight that the *overlapping* technique outperforms the *mutually-exclusive* technique. However, the differences between these miners reduce when logs contain only fully-deterministic routing decisions. This result is not surprising given the fact that the *overlapping* technique specifically focuses on discovering non-deterministic routing decision logic as included in the experiments. The largest differences in quality are with regard to recall scores in case of non-deterministic routing decisions, i.e. determinism levels equal to 0.5 and 0.75. The increase in terms of recall did not necessarily involve a tradeoff with lower precision compared to the *mutually-exclusive* technique as suggested in [71, 73].

The theoretical explanation in [71, 73] states that the *overlapping technique* only discovers different routing decision rules than the *mutually-exclusive* technique when a log contains non-deterministic routing decisions. Yet, looking at the graphs it seems that when logs contain only fully-deterministic routing decisions, the *mutually-exclusive* technique offers a small quality advantage over the *overlapping* technique. These differences, however, are not statistically significant and are caused by random differences in the samples used for the two miners.[4] We also checked these differences manually by picking logs for

[4]Recall that for each miner a separate sample from the same population is drawn.

which the *overlapping technique* had imperfect quality scores and also applied the *mutually-exclusive* technique. Both algorithms yield the same quality scores when using the same logs with fully-deterministic routing decisions. These findings indeed confirm the theory in [71, 73].

Finally, there are some extensions to the current experiments which can be explored in future empirical evaluations. One extension involves the generation of models that have other types of routing decisions than exclusive choice, such as “or” and loop. Also, one could add noise to the training log to test whether decision miners can effectively distinguish between noisy and real behavior when discovering routing decision rules. Furthermore, one could leave out certain case attributes that explain the routing decisions in the underlying process to test whether the decision mining techniques can deal with such incomplete behavior. A final possible extension relates to investigating the parameter sensitivity of decision mining algorithms. The tested algorithms can be customized by setting the values of certain parameters such as “minimum instances” that influences the level of detail and also the quality of the discovered routing decision rules.

5.2.4 Limitations and threats to validity

A possible limitation or threat to the validity of the results of this experiment are the missing alignments during quality measurement that occurred in 23% of all the folds of the cross-validation. A mismatch between the case attributes generated by the“DataExtend” method and the interior handling of those attributes by the decision mining techniques causes the missing alignments. The case attributes introduced by “DataExtend” do not change during the execution, i.e. no event changes the values of a case attribute. The tested decision mining techniques, however, operate at the event level rather than the case level. First, these techniques extend a given Petri net with write operations based on the attributes linked to events in the log. A transition t in a Petri net writes an attribute v if, according to the log, at least 66%[5] of the events related to t contain a value assignment to attribute v. In a next step, the decision miners discover the “guards” (routing decision rules) of a Data Petri net that read the previously

[5]This is the percentage defined in the default decision mining parameter settings.

Case ID	**Event ID**	**Activity**	**Z_1**
1	1	b	True
1	2	e	True
1	3	f	True
1	4	c	True
2	5	c	False
2	6	d	False

Table 5.14: Example cases for missing alignment illustration.

written attributes during replay, i.e. during alignment computation. If a routing decision appears at the very beginning of the process, then the discovered "guards" are evaluated containing attributes that have not been written before by other transitions in the Data Petri net. As a consequence, the computation of the alignment fails.

To illustrate this, consider the Data Petri net discovered by the *overlapping* technique in Figure 5.9 and two example cases in Table 5.14. The Data Petri net contains the attribute Z_1 (yellow hexagon) that is written by every visible transition in the net. The silent transition (with the double border) and transition d have guards that contain attribute Z_1. For case 1 in Table 5.14 there is a missing alignment as the "guard" $Z_1 == \mathit{False}$ tries to read attribute Z_1 which is not written before. The second case has an alignment because attribute Z_1 is written first by activity c before it is read by the guard of activity d. In our experiments, we know that the case attributes never change during execution. Therefore, one could introduce an artificial start transition that writes all the attributes of a case to prevent missing alignments. However, in reality, the case attributes can change during execution. This could make the workaround of adding an artificial transition incorrect as it is not known a priori whether the attributes are given at the beginning of the case or changed by the first activity. This illustrates that the tested decision mining algorithms could be improved to handle case attributes that do not change at the event level.

The effect of the missing alignments on the quality metrics recall, precision, and F_1 is hard to quantify. The is because we only know if an alignment of

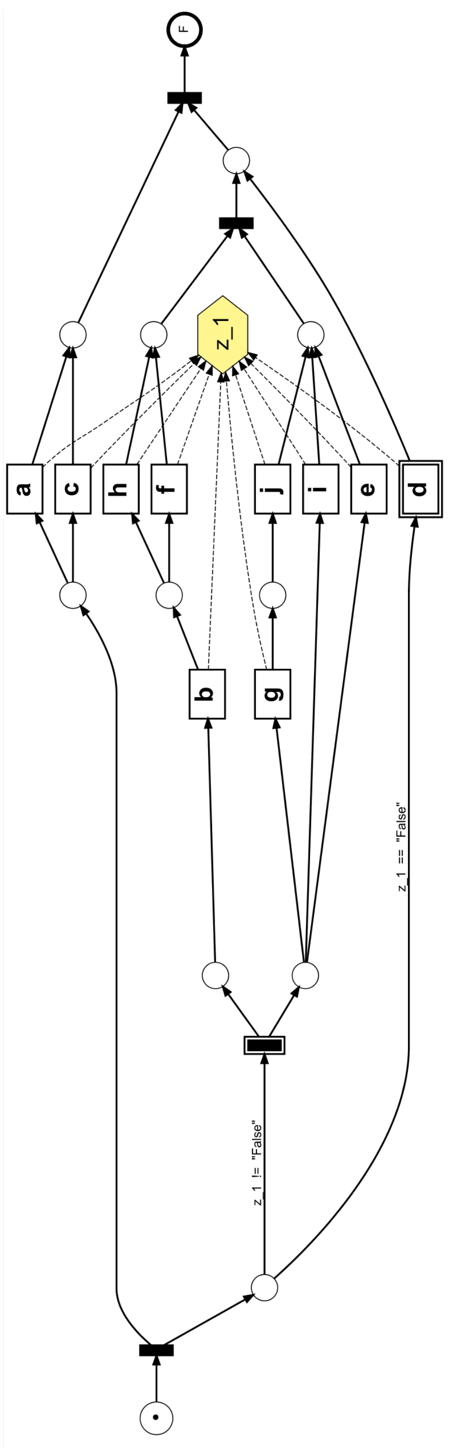


Figure 5.9: Example of discovered Data Petri net with missing alignments.

an actual fitting case is missing, an alignment of an actual non-fitting case is missing, or alignments of both actual fitting and non-fitting cases are missing. This information does not allow us to derive how many True positives, False Positives, False Negatives, or True Negatives are actually missing, and hence how much this affects recall, precision, and F_1 scores. To give an example, consider a test log with 20 cases: 10 fitting and 10 non-fitting cases. Table 5.15 illustrates the effect on recall and precision of missing alignments of non-fitting cases, missing alignments of fitting cases, and missing alignments of both fitting and non-fitting cases. In the first situation, 10 alignments were computed for the fitting cases leading to 10 True Positives and 0 False Negatives which makes recall= 1. Only 5 alignments were computed for the non-fitting cases, i.e. 5 False positives and 0 True negatives which makes precision= $\frac{2}{3}$. When there are False Positives missing, the actual precision score is lower. In the second situation, only 5 alignments were computed for the fitting cases leading to 5 True Positives and 0 False Negatives which makes recall= 1. However, if False Negatives are missing, the actual recall score is lower. Also, the current precision score is actually higher if True Positives are missing. In the last situation, only 5 and 9 alignments were computed for the fitting and non-fitting cases respectively. Many possible effects are possible on both recall and precision depending if True positives, False Negatives, or False Positives are missing.

Another limitation in the current experiment is the introduction of infrequent paths that does not guarantee an imbalance in the branches activated at each routing decision. Business processes are often characterized by exceptional paths. Additionally, it is known that imbalanced distributions of classes (here branches) provide challenges for classification approaches such as decision trees [52]. Therefore, evaluating decision miners on logs with guaranteed infrequent paths for routing decisions influenced by case attributes provides an interesting future research opportunity.

Finally, a possible threat to the validity of the experiment is that it is based on only two decision mining algorithms. As such, similar to the experiments for process discovery in the previous chapter, the experiment does not serve as a benchmark of all state-of-the-art techniques. Also, the proposed procedure is the first procedure for decision miners, the evaluation done in the experiment is

Missing non-fitting cases			
Fitting	True Pos.: 10	False Neg.: 0	Total: 10
Non-fitting	False Pos.: 5	True Neg.: 0	Total: 5
Recall	$= \frac{10}{10+0} = 1$		
Precision	$= \frac{10}{10+5} = \frac{2}{3}$	lower if FP are missing	
Missing fitting cases			
Fitting	True Pos.: 5	False Neg.: 0	Total: 5
Non-fitting	False Pos.: 5	True Neg.: 5	Total: 10
Recall	$= \frac{5}{5+0} = 1$	lower if FN are missing	
Precision	$= \frac{5}{5+5} = 0.5$	higher if TP are missing	
Missing fitting and non-fitting cases			
Fitting	True Pos.: 3	False Neg.: 2	Total: 5
Non-fitting	False Pos.: 3	True Neg.: 6	Total: 9
Recall	$= \frac{3}{3+2} = 0.6$	higher if TP are missing lower if FN are missing	
Precision	$= \frac{3}{3+3} = 0.5$	higher if TP are missing lower if FP are missing	

Table 5.15: Example missing alignments: missing non-fittin cases, missing fitting cases, and missing both fitting and non-fitting cases.

also still in the exploration phase. Future research should further empirically assess the evaluation procedure on more than two decision mining techniques. Nevertheless, the experiment is sufficient to validate the use of the evaluation procedure to benchmark decision miners and study the effect of determinism of routing decisions and infrequent paths on the quality of decision miners.

5.2.5 Requirements

Following the Design Science research principles, the evaluation assesses whether the developed artefact, i.e. the evaluation procedure for decision mining algorithms, meets its requirements. The introduction of this chapter stated that the new evaluation procedure should incorporate the GED methodology and "DataExtend" implementation as a data set selection method such that it enables extensive experiments that allow for generalization of results to a population of processes. Furthermore, the new procedure should use reference model knowledge during quality measurement to reduce the possible bias of the quality results. The new procedure meets both of these requirements by design and this was further illustrated during the experiment.

Firstly, the GED methodology and "DataExtend" implementation allow for the specification of a model population from which a (large) random sample of process models enhanced with routing decision rules and logs can be drawn. The experiment results are generalizable to that model population, taking into account the size of the event logs and the parameter settings used for the tested decision mining algorithms. We acknowledge that the "DataExtend" implementation is not complete with regard to extending all possible types of routing decisions with case attributes. Nevertheless, it provides a first and necessary step in the empirical evaluation of decision mining techniques.

Secondly, the reference model with routing decision rules is used during quality measurement. As such we can generate both fitting and non-fitting cases and label them correctly. Therefore, we can estimate the quality of the decision miner to rediscover the routing decision rules of the underlying process. In contrast, the log measures, such as the place fitness and precision metrics [71, 72], assume that the reference model with decision rules are unknown. Therefore, the metrics assume that the given log contains only fitting cases and guess the non-fitting cases. However, a given event log may be incomplete or contain noise, which could make the assumptions invalid and as a result make the quality score a biased estimate of the true quality of the decision miner to rediscover the routing decision rules of the underlying process.

5.3 Conclusion

This chapter filled an existing research gap by introducing a decision mining evaluation procedure. The procedure is an extension of the discovery evaluation procedure in Chapter 4. It starts from the GED methodology with data-flow extension called "DataExtend" (introduced in Chapter 3) to create random process models with routing decision rules and logs tailored towards empirical decision mining evaluation. In a next step, the new procedure measures the quality of the decision mining techniques to rediscover the original routing decision rules using the knowledge of the original (reference) model with routing decision rules.

An experiment including two decision mining techniques demonstrated and evaluated the proposed evaluation procedure. It was shown that the procedure allows to benchmark decision miners and analyze the impact of process characteristics, such as routing decision determinism, on miner quality. Furthermore, the evaluation has shown that the evaluation procedure overcomes the remaining challenges of data set selection and quality measurement in decision mining evaluation.

Future research opportunities include:

- introducing routing decisions rules to "or" and loop patterns,
- guarantee infrequent paths when routing decisions are influenced by case attributes,
- experiments that include noise in the training logs to make it more challenging for decision mining techniques to effectively distinguish between real behavior and behavior unrelated to the underlying process,
- experiments that input logs with incomplete case attribute information to test the robustness of the employed decision mining algorithms to rediscover the routing decision logic,
- experiments that study parameter sensitivity of decision mining techniques with regard to measured quality,
- extend the evaluation procedure to go beyond routing decisions to decisions in general, e.g. the decision on the amount of a reduction on sales price in an activity in the process, and dependencies between case attributes that

are needed to make a decision, e.g. in the form of Decision Requirement Diagrams that are induced from an event log [10, 30].

CHAPTER 6

CONCLUSIONS AND FUTURE RESEARCH

The central topic of this thesis is the empirical evaluation of both process discovery and decision mining algorithms. Although researchers have introduced many process discovery and decision mining algorithms, only recently more focus has been given to the empirical evaluation of those algorithms. Yet, such evaluations are necessary in order to reach a consensus on the quality of the available algorithms. The first chapter of this thesis introduces the unresolved challenges with regard to the evaluation of process discovery and decision mining algorithms that motivated this thesis. The second chapter provides a general overview to the field of process mining, but more importantly, elaborates on the unresolved challenges of process discovery and decision mining evaluation and the artefacts that are needed to tackle those challenges. Chapter three presents the first artefact, i.e. an artificial event data generator tailored towards both process discovery and decision mining evaluation. Chapter four incorporates the artificial data generator in a new artefact consisting of a modelling notation independent evaluation procedure for process discovery. Finally, the last artefact is a first evaluation procedure for decision mining algorithms which is introduced in chapter five.

The remainder of this section will discuss the main conclusions of this thesis followed by future research opportunities.

6.1 Main conclusions

The increasing amount of process discovery and decision mining algorithms has fueled the research on empirical evaluation of these techniques. Such an evaluation aims to provide insights on which techniques perform well on which process data, i.e. having different behavioral patterns that are relevant to the “real world”. An empirical evaluation of process discovery/decision mining techniques requires four high-level steps: determining the research objective(s), selecting the appropriate data sets, choosing a suitable performance measure, and applying the correct statistical test(s). Each of these high-level steps provide unresolved challenges that impede the research community to get insights into the strengths and weaknesses of state-of-the-art discovery/decision mining techniques. This thesis tackles the challenges of data set selection, performance measurement, and statistical tests related to process discovery. Furthemore, the thesis deals with the data set selection and performance measurement challenges related to decision mining evaluation.

The first unresolved challenge for process discovery evaluation involves the data set selection. None of the existing evaluation approaches specifies a methodology of how to select the appropriate data sets for discovery evaluation. The proposed benchmark set of real-life logs contains only 19 data sets without reference models such that they do not allow for statistically significant conclusions that can be generalized to a process population. Alternatively, existing artificial data generators are limited in the process characteristics they allow and do not guarantee a correct experimental design to ensure statistically valid conclusions. The second unresolved challenge involves the quality measurement of discovered process models. The reliance on modelling notation dependent metrics can lead to biased quality results due to erroneous conversions from one model notation to another or because certain metrics treat behaviorally equivalent modelling constructs differently. The final challenge involves the statistical tests that are needed to draw general conclusions based on the evaluation results. Current evaluation approaches have insufficiently focused on the correct experimental design of evaluation experiments: input event logs are non-random samples or from an unknown population and/or quality measurements are sensitive to bias. As a result, the evaluation results cannot be generalized to a process population.

The main unresolved challenge for decision mining evaluation concerns the lack of a standard evaluation procedure. This has resulted in few empirical evaluations of decision mining algorithms that face unresolved challenges with regard to data set selection and quality measurement during evaluation. Most evaluations have used small non-random samples of real-life or articial data sets that do not allow for statistically significant conclusions that are generalizable to a larger process population. Existing artificial data generators are not specifically tailored towards decision mining evaluation and thus provide no sufficient solution to alleviate such challenges. Furthermore, existing evaluations have applied different methods to quantify the quality of decision mining results. Similar to process discovery evaluation, most employed quality metrics are sensitive to bias. Finally, due to the restriction to small samples, no statistical analyses are incorporated in existing evaluations.

As long as the research community cannot overcome the remaining challenges of process discovery and decision mining evaluation, there will be no consensus on the quality of the available techniques. Therefore, the *main research objective of the thesis is to design empirical evaluation procedures for both process discovery and decision mining that enable objective comparison and generalization of results*. This main objective is divided in three research goals.

Firstly, this thesis presented the *Generating artificial Event Data (GED) methodology* as a general methodology for the generation of random process models and event logs for empirical evaluation of process discovery and decision mining techniques. The GED methodology starts from a definition of a process model population. Such a definition specifies which control-flow and data-flow patterns characterize the models in it and assign probabilities to them to control their occurrences in these models. Then, a random sample of process models is drawn from the population which can be simulated into a random sample of event logs. The full control over the process characteristics and the randomness of the samples enables the generalization of evaluation results to the model population. The *Process Tree and Log Generator* ("PTandLogGenerator") provides the necessary algorithms with tool support to implement the GED methodology for process discovery evaluation. These algorithms allow for process behavior such as long-term dependencies, “or” patterns, and reoccurring (duplicated) activities which are not supported by existing artificial event data generators. The evaluation of the “PTandLogGenerator” shows that it effectively supports the GED methodology and that the additional process behavior can lead to new insights into discovery algorithms. Furthermore, the algorithms of *“DataExtend”* enable the inclusion of case attributes in the routing decisions of the generated models and logs. As such it allows to generate artificial models and logs that are tailored for decision mining evaluation. The evaluation of “DataExtend” has illustrated that it allows to control for the routing decision logic which is necessary while testing decision mining techniques.

Secondly, the thesis incorporates the GED methodology into a *new evaluation procedure for process discovery algorithms*. This new procedure focuses on measuring the quality of a discovery algorithm’s ability to rediscover the underlying process independently from the algorithm’s modelling notation. It starts from a user defined model population from which random reference models and logs are drawn. Then, it measures the quality of a discovery algorithm by taking a classification approach while using the knowledge of the generated reference models. Two rounds of experiments on four discovery algorithms that use different modelling notations have shown that the new procedure effectively supports the objectives of empirical process discovery evaluation: benchmarking and assessing the impact of process characteristics on discovery algorithm quality. Furthermore, the findings of the statistical analyses can be generalized to the user defined model populations. Finally, the tool implementation of the new procedure enables researchers to automate their evaluation experiments and share or extend their experiment setups to improve their reproducibility.

Finally, the thesis extends the newly proposed discovery evaluation procedure to introduce the *first evaluation procedure for decision mining techniques*. This new procedure also integrates the GED methodology to define a model population and then generate random reference models and event logs with routing decisions that depend on case attributes. In a next step, the new procedure measures the decision miner’s quality to rediscover the routing decision logic of the underlying process. The quality measurement applies a classification approach that exploits the knowledge of the known reference model. Experiments have shown that the new procedure enables researchers to benchmark decision mining algorithms and study the impact of process characteristics such as the determinism of routing decisions on decision miner's quality. Moreover, due to the experimental design that starts from a model population definition, the experiment results can be generalized to that specific population.

Overall, with these new empirical evaluation procedures, this thesis aims to stimulate more evaluation experiments and fuel the research on empirically comparing process discovery and decision mining algorithms. First of all, the new evaluation procedures support benchmarking to help researches measure the true quality ratios between different state-of-the-art algorithms. This provides answers to questions as to which process discovery algorithm performs best on event logs containing process behavior that is challenging to rediscover, e.g. reoccurring (duplicated) activities. Answers to such questions help researchers in assessing the quality improvements of a new algorithm over currently available algorithms. Secondly, the new evaluation procedures support sensitivity analysis such that researchers can accurately assess the impact of certain process behavior on the quality of the evaluated algorithm, e.g. the determinism of routing decisions on decision miner's quality. Such assessments are vital to get a detailed understanding of the empirical workings of process discovery or decision mining algorithms. Based on those insights strengths and weaknesses of algorithms are identified such that they can assist researchers in finetuning both existing and newly developed algorithms. Finally, the gained knowledge of both benchmarking and sensitivity analysis assist the process mining research domain to formulate recommendations on how to choose the most suitable discovery/decision mining technique in practice.

This thesis focuses on empirical evaluation rather than theoretical evaluations of algorithms (e.g. see [110]). However, both types of evaluations are needed as they can complement each other, e.g. an empirical analysis can reveal that the quality of a certain algorithm decreases in the presence of particular process behavior which can be further explained by a theoretical analysis of that algorithm. Another important caveat regards the fact that the evaluation procedures concentrate on artificial event logs. This does not mean that empirical evaluation should only consider artificial event logs. On the contrary, when testing the applicability of process discovery and decision mining algorithms in practice, real-life event logs are needed. Testing the scalability of the algorithms and the usefulness of the discovered models and decision rules are some of the issues that need to be investigated with real-life logs. A final remark concerns the quality measurement of the proposed evaluation procedures. In this thesis, we have measured the quality of the discovered models and routing decision rules by focusing on their behavior rather then their structure. However, the structure is typically used to measure the complexity of the discovered models and rules, i.e. another important quality dimension to consider when comparing algorithms. An algorithm may deliberately decrease the recall and precision quality of the discovered model or rules to increase their simplicity. The structure of the model (possibly extended with routing decision rules) is heavily connected to the modelling notation applied by the process discovery/decision mining algorithm. As a consequence, we have dropped the complexity quality dimension when targetting modelling notation independent quality measurement. This highlights an important limitation of the proposed evaluation procedures that requires more research on comparing the complexity of models (with rules) in different notations to solve it.

6.2 Future research opportunities

A significant number of future research opportunities have been identified in the previous chapters. Those research opportunities mainly focus on the developed algorithms and evaluation procedures. This section summarizes the key challenges and describes several more general research directions.

With respect to the “PTandLogGenerator” and “DataExtend” implementations of the GED methodology, future work can extend the process behavior that the generated models (with routing decision rules) and logs can contain. Firstly, the long-term dependencies introduced by the “PTandLogGenerator” are limited to exclusive choices. Future work could extend these dependencies to non-exclusive choices (“or” pattern). Similarly, “DataExtend” restricts the introduction of case attributes to routing decisions involving exclusive choices. Here, future work could also consider to extend loops and “or” with case attributes. Both the long-term dependencies and routing decision extensions would provide interesting challenges for state-of-the-art process discovery and decision mining algorithms respectively. Thirdly, the current log characteristics include only the number of cases and the amount of noise. These log characteristics should be extended, e.g. using the log metrics indrocuded by Günther [41]. Finally, future extensions of “PTandLogGenerator” and “DataExtend” should not be restricted to process trees, but could adopt richer modeling notations such as BPMN [46].

A first possible extension to the new process discovery evaluation procedure is the introduction of noise in the training logs. This would make it more difficult for an algorithm to rediscover the underlying process. Another possible extension relates to the noise operations that are used to punish imprecise behavior in the discovered models. Such an extension could solve the current inability to punish the non-detection of long-term dependencies. A third extension regards the optimization of the parameter values of the evaluated process discovery algorithms in order to maximize their real capabilities. Finally, the RapidProM tool implementation should be extended to include more discovery algorithms using other modelling notations such as BPMN to enable for empirical evaluations that cover all state-of-the-art techniques.

Also several extensions for the new decision mining evaluation procedure are worthwhile to explore. The first extension is to control for imbalanced execution of routing decisions influenced by case attributes. In this way, one can test how such imbalances impact the quality of the decision miners that apply classification approaches to rediscover the routing decision rules. Another extension involves deliberately removing or adapting some of the case attribute information to test the robustness of decision miners to deal with noise in the training log. A final extension is to expand the decision mining evaluation procedure to go beyond routing decisions to more general decisions that are taken during process execution. Also, evaluating the quality of discovered Decision Requirement Diagrams would present an interesting future research opportunity.

Except from the above research opportunities, also more general research directions are identified. Firstly, more research is needed to determine the process population from which real event logs come. If one can learn the population parameters, then one can generate larger samples (of artificial event logs) from those populations. That would allow researchers to evaluate on these larger samples and generalize the results to the real-life process population. Secondly, the new evaluation procedures in this thesis focus on two process mining tasks, i.e. process discovery and decision mining, but these procedures could be extended to other tasks as well. Example extensions involve algorithms that mine organizational process models and provide time overlay. Thirdly, empirical analysis should become part of the development of new process discovery algorithms. Currently, there exists a strong tendency to build new algorithms based on novel approaches rather than building on previously generated algorithms. However, insights on the empirical workings of current algorithms provide best practices which can be used to improve the state of the art. To fill this gap, research is needed which uncovers and explains successes and flaws in current process discovery techniques and how to use these to achieve better algorithms. Finally, offering empirical evaluation procedures as web services could further improve the usability and scalability of empirical analysis. For example, researchers are given a random sample of event logs to which they apply their algorithm. Next, the researchers upload their discovered models and the web service evaluates them and returns the evaluation results. Such services ensure that algorithms are evaluated in a standard way and offload the computational capacity that a researcher needs for the evaluation.

APPENDIX A

APPENDIX A

Traces
⟨ issue order, produce order, inspect thoroughly, package, deliver, send invoice ⟩^{18}
⟨ issue order, produce order, inspect normally, inspect thoroughly, package, deliver, send invoice ⟩^{15}
⟨ issue order, produce order, inspect thoroughly, inspect normally, package, deliver, send invoice ⟩^{15}
⟨ issue order, produce order, inspect thoroughly, package, send invoice, deliver ⟩^{14}
⟨ issue order, produce order, inspect normally, package, send invoice, deliver ⟩^{14}
⟨ issue order, produce order, inspect thoroughly, inspect normally, package, send invoice, deliver ⟩^{10}
⟨ issue order, produce order, inspect normally, inspect thoroughly, package, send invoicedeliver ⟩^{8}
⟨ issue order, produce order, inspect normally, package, deliver, send invoice ⟩^{6}

Table A.1: Example event log about make-to-order process with “or” pattern, L_{or}.

APPENDIX B

Please refer to the tables from Table B.1 to B.11

Kruskall-Wallis rank sum test			
KW $\chi^2 = 2303.8$	degrees of freedom = 3		p-value $< 2.2e^{-16}$
Multiple comparison test after Kruskall-Wallis ($\alpha = 0.05$)			
Comparisons	*Observed diff.*	*Critical diff.*	*Significant diff.*
Alpha+ - Heuristics	328.3	126.9	True
Alpha+ - ILP	1621.1	126.9	True
Alpha+ - Inductive	1033.7	126.9	True
Heuristics - ILP	1949.4	126.9	True
Heuristics - Inductive	1362	126.9	True
ILP - Inductive	587.4	126.9	True

Table B.1: Results of the statistical tests to study the effect of discovery algorithm on F_1 scores.

Kruskall-Wallis rank sum test			
KW $\chi^2 = 876.57$	degrees of freedom = 3		p-value $< 2.2e^{-16}$
Multiple comparison test after Kruskall-Wallis ($\alpha = 0.05$)			
Comparisons	*Observed diff.*	*Critical diff.*	*Significant diff.*
Alpha+ - Heuristics	217.78	89.78	True
Alpha+ - ILP	565.52	89.78	True
Alpha+ - Inductive	532.52	89.78	True
Heuristics - ILP	783.29	89.78	True
Heuristics - Inductive	750.30	89.78	True
ILP - Inductive	32.99	89.78	False

Table B.2: Results of the statistical tests to study the effect of infrequent behavior on precision scores.

Comparisons	Observed diff.	Critical diff.	Significant diff.
Kruskall-Wallis rank sum test			
KW $\chi^2 = 44.29$	degrees of freedom = 6	p-value $< 6.485e^{-8}$	
Jonckheere-Terpstra test			
JT = 140560	p-value = 0.002		
Multiple comparison test after Kruskall-Wallis ($\alpha = 0.05$)			
Comparisons	*Observed diff.*	*Critical diff.*	*Significant diff.*
0.0-0.05	78.10	96.73	False
0.0-0.1	88.40	96.73	False
0.0-0.15	111.32	96.73	**True**
0.0-0.2	96.23	96.73	False
0.0-0.25	139.90	96.73	**True**
0.0-0.3	151.32	96.73	**True**
0.05-0.1	10.30	96.73	False
0.05-0.15	33.22	96.73	False
0.05-0.2	18.13	96.73	False
0.05-0.25	61.80	96.73	False
0.05-0.3	73.21	96.73	False
0.1-0.15	22.92	96.73	False
0.1-0.2	7.83	96.73	False
0.1-0.25	51.50	96.73	False
0.1-0.3	62.92	96.73	False
0.15-0.2	15.09	96.73	False
0.15-0.25	28.58	96.73	False
0.15-0.3	40.0	96.73	False
0.2-0.25	43.67	96.73	False
0.2-0.3	55.09	96.73	False
0.25-0.3	11.42	96.73	False

Table B.3: Results of the statistical tests to study the effect of reoccurring activities on F_1 scores for the Alpha+ miner.

Kruskall-Wallis rank sum test		
KW $\chi^2 = 1.4786$	degrees of freedom = 6	p-value = 0.9609
Jonckheere-Terpstra test		
JT = 160160	p-value = 0.496	

Table B.4: Results of the statistical tests to study the effect of reoccurring activities on F_1 scores for the Heuristics miner.

Kruskall-Wallis rank sum test			
KW $\chi^2 = 331.81$	degrees of freedom = 6	p-value $< 2.2e^{-16}$	
Jonckheere-Terpstra test			
JT = 81029		p-value = 0.002	
Multiple comparison test after Kruskall-Wallis ($\alpha = 0.05$)			
Comparisons	*Observed diff.*	*Critical diff.*	*Significant diff.*
0.0-0.05	103.28	96.73	True
0.0-0.1	184.87	96.73	True
0.0-0.15	244.71	96.73	True
0.0-0.2	333.78	96.73	True
0.0-0.25	416.40	96.73	True
0.0-0.3	459.0	96.73	True
0.05-0.1	81.59	96.73	False
0.05-0.15	141.42	96.73	True
0.05-0.2	230.50	96.73	True
0.05-0.25	313.12	96.73	True
0.05-0.3	355.72	96.73	True
0.1-0.15	59.83	96.73	False
0.1-0.2	148.91	96.73	True
0.1-0.25	231.53	96.73	True
0.1-0.3	274.13	96.73	True
0.15-0.2	89.07	96.73	False
0.15-0.25	171.70	96.73	True
0.15-0.3	214.29	96.73	True
0.2-0.25	82.63	96.73	False
0.2-0.3	125.22	96.73	True
0.25-0.3	42.60	96.73	False

Table B.5: Results of the statistical tests to study the effect of reoccurring activities on F_1 scores for the ILP Miner.

Kruskall-Wallis rank sum test			
KW $\chi^2 = 180$	degrees of freedom = 6	p-value $< 2.2e^{-16}$	
Jonckheere-Terpstra test			
JT = 105510	p-value = 0.002		
Multiple comparison test after Kruskall-Wallis ($\alpha = 0.05$)			
Comparisons	*Observed diff.*	*Critical diff.*	*Significant diff.*
0.0-0.05	119.75	96.73	True
0.0-0.1	190.98	96.73	True
0.0-0.15	255.02	96.73	True
0.0-0.2	281.17	96.73	True
0.0-0.25	286.07	96.73	True
0.0-0.3	370.57	96.73	True
0.05-0.1	71.24	96.73	False
0.05-0.15	135.27	96.73	True
0.05-0.2	161.43	96.73	True
0.05-0.25	166.32	96.73	True
0.05-0.3	250.82	96.73	True
0.1-0.15	64.04	96.73	False
0.1-0.2	90.19	96.73	False
0.1-0.25	95.08	96.73	False
0.1-0.3	179.58	96.73	True
0.15-0.2	26.15	96.73	False
0.15-0.25	31.05	96.73	False
0.15-0.3	115.55	96.73	True
0.2-0.25	4.90	96.73	False
0.2-0.3	89.40	96.73	False
0.25-0.3	84.50	96.73	False

Table B.6: Results of the statistical tests to study the effect of reoccurring activities on F_1 scores for the Inductive miner.

Kruskall-Wallis rank sum test			
Recall	KW $\chi^2 = 4.09$	degrees of freedom = 1	p-value = 0.0432
Precision	KW $\chi^2 = 0.04$	degrees of freedom = 1	p-value = 0.8483
F_1	KW $\chi^2 = 1.80$	degrees of freedom = 1	p-value = 0.1795

Table B.7: Results of the statistical tests to study the effect of decision mining algorithm on recall, precision, and F_1 scores.

Kruskall-Wallis rank sum test			
Recall	KW $\chi^2 = 1.21$	degrees of freedom = 1	p-value = 0.27
Precision	KW $\chi^2 = 1.02$	degrees of freedom = 1	p-value = 0.31
F_1	KW $\chi^2 = 2.57$	degrees of freedom = 1	p-value = 0.11

Table B.8: Results of the statistical tests to study the effect of infrequent behaviour on recall, precision, and F_1 scores.

Kruskall-Wallis rank sum test			
Recall	KW $\chi^2 = 3.06$	degrees of freedom = 1	p-value = 0.08
Precision	KW $\chi^2 = 0.55$	degrees of freedom = 1	p-value = 0.46
F_1	KW $\chi^2 = 0.07$	degrees of freedom = 1	p-value = 0.79

Table B.9: Results of the statistical tests to study the effect of absence of infrequent behavior on recall, precision, and F_1 scores.

Kruskall-Wallis rank sum test			
KW $\chi^2 = 18.749$	degrees of freedom = 2	p-value = $8.486e-05$	
Jonckheere-Terpstra test			
JT = 113780		p-value = 0.002	
Multiple comparison test after Kruskall-Wallis ($\alpha = 0.05$)			
Comparisons	*Observed diff.*	*Critical diff.*	*Significant diff.*
0.5-0.75	20.69961	47.12554	False
0.5-1	80.91085	47.12554	True
0.75-1	60.21124	47.12554	True

Table B.10: Results of the statistical tests to study the effect of determinism level on F_1 scores for the mutually-exclusive technique.

Kruskall-Wallis rank sum test			
KW χ^2 = 11.067	degrees of freedom = 2	p-value = 0.003952	
Jonckheere-Terpstra test			
JT = 110860		p-value = 0.002	
Multiple comparison test after Kruskall-Wallis (α = 0.05)			
Comparisons	*Observed diff.*	*Critical diff.*	*Significant diff.*
0.5-0.75	41.35659	47.12554	False
0.5-1	63.43992	47.12554	True
0.75-1	22.08333	47.12554	False

Table B.11: Results of the statistical tests to study the effect of determinism level on F_1 scores for the overlapping technique.

APPENDIX C

This appendix includes additional definitions adopted from [16].

Definition C.1 (Projection)**.** Let A be a set of elements, then for all $A' \subseteq A$, $\sigma_{\downarrow A'}$ denotes the projection of a sequence $\sigma \in A^*$ on A', e.g. $\langle a,a,b,c\rangle_{\downarrow\{a,c\}} = \langle a,a,c\rangle$.

Definition C.2 (Process tree language)**.** Let $A \subseteq \mathscr{A}$ be a set of activities, let $PT = (N,r,m,c,p,b)$ be a process tree, and let $\sigma \cdot \sigma'$ denote the concatenation of two sequences. The language of a process tree $\mathscr{L} : N \to A^*$ is defined as the language of the root note $\mathscr{L}(r)$. The language of a node n in a process tree is defined as follows:

- if $m(n) = \tau$, then $\mathscr{L}(n) = \{\langle\rangle\}$
- if $m(n) = a \in A$, then $\mathscr{L}(n) = \{\langle a\rangle\}$
- if $m(n) \in O$ and $c(n) = \langle n_1,\ldots,n_k\rangle$, then
  - if $m(n) = \to$, then $\mathscr{L}(n) = \{\sigma | \exists \sigma_1 \in \mathscr{L}(n_1)\ldots\sigma_k \in \mathscr{L}(n_k) : \sigma = \sigma_1 \cdot \ldots \cdot \sigma_k\}$
  - if $m(n) = \times$, then $\mathscr{L}(n) = \{\sigma | \exists 1 \leq i \leq k : \sigma_i \in \mathscr{L}(n_i)\} = \bigcup_{1\leq i\leq k} \mathscr{L}(n_i)$
  - if $m(n) = \vee$, then $\mathscr{L}(n) = \{\sigma \in A^* | \sigma = \langle\rangle \Rightarrow (\exists n' \in c(n) : \langle\rangle \in \mathscr{L}(n')) \wedge \sigma \neq \langle\rangle \Rightarrow (\exists f : \{1\ldots|\sigma|\} \to c(n) : \forall n' \in Rng(f) : \sigma_{\downarrow n'} \in \mathscr{L}(n'))\}$

- if $m(n) = \wedge$, then $\mathscr{L}(n) = \{\sigma \in A^* | \sigma = \langle\rangle \Rightarrow \forall n' \in c(n) : \langle\rangle \in \mathscr{L}(n') \wedge \sigma \neq \langle\rangle \Rightarrow (\exists f : \{1 \dots |\sigma|\} \rightarrow c(n) : \forall n' \in Rng(f) : \sigma_{\downarrow n'} \in \mathscr{L}(n') \wedge \forall n' \in c(n) \setminus Rng(f) : \langle\rangle \in \mathscr{L}(n'))\}$
- if $m(n) = ↺^k$, then $\mathscr{L}(n) = \{\sigma_1 \cdot \sigma_2 \cdot \sigma_3 \in A^* | \sigma_1 \in \mathscr{L}(c(n)_1) \wedge \sigma_3 \in \mathscr{L}(c(n)_3) \wedge \sigma_2 \in f(c(n)_2, c(n)_1)\}$ with $f : N \times N \rightarrow A^* : f(n_1, n_2) = \{\sigma | \sigma = \langle\rangle \vee (\sigma = \sigma_1 \cdot \sigma_2 \cdot \sigma_3 \in A^* \wedge \sigma_1 \in \mathscr{L}(c(n)_1) \wedge \sigma_2 \in \mathscr{L}(c(n)_2) \wedge \sigma_3 \in f(n_1, n_2))\}$. Notice that the maximum number of iterations k forces $\sigma = \langle\rangle$ in function $f(n_1, n_2)$ after k iterations of $\sigma = \sigma_1 \cdot \sigma_2 \cdot \sigma_3$.

To illustrate the language of the different operators, consider the following examples:

- $\mathscr{L}(\rightarrow (a, b)) = \{\langle a, b\rangle\}$
- $\mathscr{L}(\times(a, b)) = \{\langle a\rangle, \langle b\rangle\}$
- $\mathscr{L}(\wedge(a, b)) = \{\langle a, b\rangle, \langle b, a\rangle\}$
- $\mathscr{L}(\vee(a, b)) = \{\langle a\rangle, \langle b\rangle, \langle a, b\rangle, \langle b, a\rangle\}$
- $\mathscr{L}(↺^2 (a, b, c)) = \{\langle a, c\rangle, \langle a, b, a, c\rangle, \langle a, b, a, b, a, c\rangle\}$

BIBLIOGRAPHY

[1] *IEEE Standard for eXtensible Event Stream (XES) for Achieving Interoperability in Event Logs and Event Streams*, IEEE Std 1849-2016, (2016), pp. 1–50.

[2] D. N. A. ASUNCION, *UCI machine learning repository*, 2007.

[3] A. ADRIANSYAH, J. MUNOZ-GAMA, J. CARMONA, B. F. VAN DONGEN, AND W. M. P. VAN DER AALST, *Measuring precision of modeled behavior*, Information Systems and e-Business Management, (2015), pp. 1–31.

[4] A. ADRIANSYAH, B. F. VAN DONGEN, AND W. M. P. VAN DER AALST, *Towards robust conformance checking*, in Business Process Management Workshops, vol. 66, Springer, 2011, pp. 122–133.

[5] J. ANTONAKIS, S. BENDAHAN, P. JACQUART, AND R. LALIVE, *On making causal claims: A review and recommendations*, The Leadership Quarterly, 21 (2010), pp. 1086–1120.

[6] A. AUGUSTO, R. CONFORTI, M. DUMAS, M. LA ROSA, F. M. MAGGI, A. MARRELLA, M. MECELLA, AND A. SOO, *Automated Discovery of Process Models from Event Logs: Review and Benchmark*, IEEE Transactions on Knowledge and Data Engineering, (2018).

[7] A. BARKER AND J. VAN HEMERT, *Scientific Workflow: a Survey and Research Directions*, in Parallel Processing and Applied Mathematics, Springer, 2007, pp. 746–753.

[8] K. Batoulis, S. Haarmann, and M. Weske, *Various notions of soundness for decision-aware business processes*, in International Conference on Conceptual Modeling, Springer, 2017, pp. 403–418.

[9] K. Batoulis and M. Weske, *Soundness of decision-aware business processes*, in International Conference on Business Process Management, Springer, 2017, pp. 106–124.

[10] E. Bazhenova, S. Buelow, and M. Weske, *Discovering Decision Models from Event Logs*, in International Conference on Business Information Systems, Springer, 2016, pp. 237–251.

[11] A. Benavoli, G. Corani, and F. Mangili, *Should we really use post-hoc tests based on mean-ranks*, Journal of Machine Learning Research, 17 (2016), pp. 1–10.

[12] M. A. Bender, M. Farach-Colton, G. Pemmasani, S. Skiena, and P. Sumazin, *Lowest common ancestors in trees and directed acyclic graphs*, Journal of Algorithms, 57 (2005), pp. 75–94.

[13] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser, *Process mining based on regions of languages*, in Business Process Management, G. Alonso, P. Dadam, and M. Rosemann, eds., Springer, 2007, pp. 375–383.

[14] A. Bolt, M. de Leoni, and W. M. P. van der Aalst, *Scientific Workflows for Process Mining: Building Blocks, Scenarios, and Implementation*, Software Tools for Technology Transfer, 18 (2016), p. 22.

[15] G. E. Box, J. S. Hunter, and W. G. Hunter, *Statistics for experimenters: design, innovation, and discovery*, vol. 2, Wiley-Interscience New York, 2005.

[16] J. C. A. M. Buijs, *Flexible Evolutionary Algorithms for Mining Structured Process Models*, PhD thesis, Technische Universiteit Eindhoven, Eindhoven, 2014.

[17] A. BURATTIN, *PLG2: Multiperspective Processes Randomization and Simulation for Online and Offline Settings*, tech. rep., University of Innsbruck, June 2015.

[18] ——, *Process mining techniques in business environments*, vol. 207 of Lecture Notes in Business Information Processing, Springer, Cham, 2015.

[19] ——, *PLG2: Multiperspective Process Randomization with Online and Offline Simulations.*, in BPM (Demos), 2016, pp. 1–6.

[20] A. BURATTIN AND A. SPERDUTI, *PLG: A framework for the generation of business process models and their execution logs*, in Business Process Management Workshops, Springer, 2011, pp. 214–219.

[21] D. CALVANESE, M. MONTALI, A. SYAMSIYAH, AND W. M. P. V. D. AALST, *Ontology-Driven Extraction of Event Logs from Relational Databases*, in Business Process Management Workshops, Lecture Notes in Business Information Processing, Springer, Cham, Aug. 2015, pp. 140–153.

[22] J. CARMONA, M. DE LEONI, B. DEPAIRE, AND T. JOUCK, *Summary of the Process Discovery Contest 2016*, in Business Process Management Workshops: BPM 2016 International Workshops Rio de Janeiro, Brazil, September 19, 2016, vol. 281 of Lecture Notes in Business Information Processing, Rio de Janeiro, 2017, Springer, pp. 7–10.

[23] J. COHEN, *Statistical power analysis for the behavioral sciences*, Lawrence Erlbaum Associates, Hillsdale, second ed., 1988.

[24] R. CONFORTI, M. DUMAS, L. GARCIA-BANUELOS, AND M. LA ROSA, *BPMN Miner: Automated discovery of BPMN process models with hierarchical structure*, Information Systems, 56 (2016), pp. 284–303.
WOS:000367634200017.

[25] M. DE LEONI, M. DUMAS, AND L. GARCÍA-BAÑUELOS, *Discovering branching conditions from business process execution logs*, in Interna-

tional Conference on Fundamental Approaches to Software Engineering, Springer, 2013, pp. 114–129.

[26] M. DE LEONI, P. FELLI, AND M. MONTALI, *A Holistic Approach for Soundness Verification of Decision-Aware Process Models*, tech. rep., Eindhoven University of Technology, Apr. 2018.

[27] M. DE LEONI AND W. M. VAN DER AALST, *Aligning event logs and process models for multi-perspective conformance checking: An approach based on integer linear programming*, in Business Process Management, Springer, 2013, pp. 113–129.

[28] M. DE LEONI AND W. M. P. VAN DER AALST, *Data-aware process mining: discovering decisions in processes using alignments*, in Proceedings of the 28th Annual ACM Symposium on Applied Computing, ACM, 2013, pp. 1454–1461.

[29] A. K. A. DE MEDEIROS, A. J. WEIJTERS, AND W. M. P. VAN DER AALST, *Genetic process mining: an experimental evaluation*, Data Mining and Knowledge Discovery, 14 (2007), pp. 245–304.

[30] J. DE SMEDT, F. HASIĆ, S. K. VANDEN BROUCKE, AND J. VANTHIENEN, *Towards a holistic discovery of decisions in process-aware information systems*, in International Conference on Business Process Management, Springer, 2017, pp. 183–199.

[31] J. DE WEERDT, *Business Process Discovery: New Techniques and Applications*, PhD thesis, KU Leuven, Leuven, 2012.

[32] J. DE WEERDT, M. DE BACKER, J. VANTHIENEN, AND B. BAESENS, *A robust F-measure for evaluating discovered process models*, in Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on, IEEE, 2011, pp. 148–155.

[33] ——, *A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs*, Information Systems, 37 (2012), pp. 654–676.

[34] J. DEMŠAR, *Statistical comparisons of classifiers over multiple data sets*, The Journal of Machine Learning Research, 7 (2006), pp. 1–30.

[35] C. DI CICCIO, M. L. BERNARDI, M. CIMITILE, AND F. M. MAGGI, *Generating event logs through the simulation of Declare models*, in Workshop on Enterprise and Organizational Modeling and Simulation, Springer, 2015, pp. 20–36.

[36] C. DI CICCIO AND M. MECELLA, *A two-step fast algorithm for the automated discovery of declarative workflows*, in Computational Intelligence and Data Mining (CIDM), 2013 IEEE Symposium on, IEEE, 2013, pp. 135–142.

[37] M. DUMAS, M. LA ROSA, J. MENDLING, AND H. A. REIJERS, *Fundamentals of business process management*, Springer, 2013.

[38] F. FAUL, E. ERDFELDER, A. BUCHNER, AND A.-G. LANG, *Statistical Power Analyses Using G* Power 3.1: Tests for Correlation and Regression Analyses*, Behavior research methods, 41 (2009), pp. 1149–1160.

[39] C. FAVRE, D. FAHLAND, AND H. VÖLZER, *The relationship between workflow graphs and free-choice workflow nets*, Information Systems, 47 (2015), pp. 197–219.

[40] S. R. A. FISHER, *The Design of Experiments*, vol. 12, Oliver and Boyd Edinburgh, 1960.

[41] C. W. GÜNTHER, *Process mining in flexible environments*, PhD thesis, Technische Universiteit Eindhoven, 2009.

[42] C. W. GÜNTHER AND W. M. P. VAN DER AALST, *Fuzzy mining–adaptive process simplification based on multi-perspective metrics*, in Business Process Management, Springer, 2007, pp. 328–343.
PM101 - 4.

[43] S. GOEDERTIER, D. MARTENS, J. VANTHIENEN, AND B. BAESENS, *Robust process discovery with artificial negative events*, The Journal of Machine Learning Research, 10 (2009), pp. 1305–1340.

[44] I. GOODFELLOW, Y. BENGIO, AND A. COURVILLE, *Deep Learning*, MIT Press, 2016.
`http://www.deeplearningbook.org`.

[45] A. G. GREENWALD, *Within-subjects designs: To use or not to use?*, Psychological Bulletin, 83 (1976), p. 314.

[46] O. M. GROUP, *Business Process Model and Notation specification*, Jan. 2011.

[47] ——, *Decision Model and Notation specification*, 2016.

[48] M. JANS AND P. SOFFER, *From relational database to event log: decisions with quality impact*, in International Conference on Business Process Management, Springer, 2017, pp. 588–599.

[49] G. JANSSENSWILLEN, B. DEPAIRE, AND T. JOUCK, *Calculating the number of unique paths in a block-structured process model*, in Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data 2016, 2016.

[50] G. JANSSENSWILLEN, T. JOUCK, M. CREEMERS, AND B. DEPAIRE, *Measuring the Quality of Models with Respect to the Underlying System: An Empirical Study*, in Business Process Management, M. L. Rosa, P. Loos, and O. Pastor, eds., no. 9850 in Lecture Notes in Computer Science, Springer International Publishing, Sept. 2016, pp. 73–89.

[51] N. JAPKOWICZ AND M. SHAH, *Evaluating learning algorithms: a classification perspective*, Cambridge University Press, 2011.

[52] N. JAPKOWICZ AND S. STEPHEN, *The class imbalance problem: A systematic study*, Intelligent data analysis, 6 (2002), pp. 429–449.

[53] K. JENSEN, L. M. KRISTENSEN, AND L. WELLS, *Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems*, International Journal on Software Tools for Technology Transfer, 9 (2007), pp. 213–254.

[54] T. JIN, J. WANG, AND L. WEN, *Efficiently Querying Business Process Models with BeehiveZ.*, in BPM (Demos), 2011.

[55] P. JOHANNESSON AND E. PERJONS, *An Introduction to Design Science*, Springer, 2014.

[56] T. JOUCK, A. BOLT, B. DEPAIRE, M. DE LEONI, AND W. M. P. VAN DER AALST, *An Integrated Framework for Process Discovery Algorithm Evaluation*, Business & Information Systems Engineering, Submitted, 1st review (2018).

[57] T. JOUCK, M. DE LEONI, AND B. DEPAIRE, *Generating Decision-aware Models & Logs: Towards an Evaluation of Decision Mining*, in Proceedings of the 6th International Workshop on Declarative/Decision/Hybrid Mining and Modelling for Business Processes (DeHMiMoP 2018), Sydney, June 2018, Springer, p. 12.

[58] T. JOUCK AND B. DEPAIRE, *Generating Artificial Event Logs with Sufficient Discriminatory Power to Compare Process Discovery Techniques*, in Proceedings of the 4th International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2014), CEUR Workshop Proceedings, 2014, pp. 174–178.

[59] ——, *PTandLogGenerator: a Generator for Artificial Event Data*, in Proceedings of the BPM Demo Track 2016 (BPMD 2016), vol. 1789, Rio de Janeiro, 2016, CEUR workshop proceedings, pp. 23–27.

[60] ——, *Simulating Process Trees Using Discrete-Event Simulation*, technical Report, Hasselt University, Feb. 2017.

[61] ——, *Generating Artificial Data for Empirical Analysis of Control-flow Discovery Algorithms: A Process Tree and Log Generator*, Business & Information Systems Engineering, Accepted (2018).

[62] A. KALENKOVA, M. DE LEONI, AND W. M. P. VAN DER AALST, *Discovering, analyzing and enhancing BPMN models using prom*, 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., vol. 1295 of CEUR Workshop Proceedings, CEUR-WS.org, 2014, p. 36.

[63] V. KATAEVA, R. F. MOSCOW, AND A. A. KALENKOVA, *Applying graph grammars for the generation of process models and their logs*, in Proceedings of the Spring/Summer Young Researchers' Colloquium on Software Engineering., 2014.

[64] R. E. KIRK, *Experimental Design*, Wiley Online Library, 1982.

[65] M. KUNZE, A. LUEBBE, M. WEIDLICH, AND M. WESKE, *Towards Understanding Process Modeling — the Case of the BPM Academic Initiative*, in International Workshop on Business Process Modeling Notation, Springer, 2011, pp. 44–58.

[66] M. LEEMANS AND W. M. P. VAN DER AALST, *Discovery of frequent episodes in event logs*, in International Symposium on Data-Driven Process Discovery and Analysis, Springer, 2014, pp. 1–31.

[67] S. J. LEEMANS, D. FAHLAND, AND W. M. P. VAN DER AALST, *Discovering block-structured process models from event logs containing infrequent behaviour*, in Business Process Management Workshops, Springer, 2014, pp. 66–78.

[68] S. J. J. LEEMANS, *Robust process mining with guarantees*, PhD thesis, Ph. D. thesis, Eindhoven University of Technology, 2017.

[69] S. J. J. LEEMANS, D. FAHLAND, AND W. M. P. VAN DER AALST, *Discovering block-structured process models from event logs-a constructive approach*, in Application and Theory of Petri Nets and Concurrency, Springer, 2013, pp. 311–329.

[70] X. LU, D. FAHLAND, F. J. H. M. V. D. BIGGELAAR, AND W. M. P. VAN DER AALST, *Handling Duplicated Tasks in Process Discovery by Refining Event Labels*, in Business Process Management, M. L. Rosa, P. Loos,

and O. Pastor, eds., no. 9850 in Lecture Notes in Computer Science, Springer International Publishing, Sept. 2016, pp. 90–107.

[71] F. MANNHARDT, *Multi-perspective process mining*, PhD thesis, Technische Universiteit Eindhoven, Eindhoven, 2018.

[72] F. MANNHARDT, M. DE LEONI, H. A. REIJERS, AND W. M. VAN DER AALST, *Measuring the precision of multi-perspective process models*, in International Conference on Business Process Management, Springer, 2015, pp. 113–125.

[73] F. MANNHARDT, M. DE LEONI, H. A. REIJERS, AND W. M. P. VAN DER AALST, *Balanced multi-perspective checking of process conformance*, Computing, 98 (2016), pp. 407–437.

[74] ——, *Decision Mining Revisited-Discovering Overlapping Rules.*, in CAiSE, vol. 9694, 2016, pp. 377–392.

[75] T. MCPHILLIPS, S. BOWERS, D. ZINN, AND B. LUDÄSCHER, *Scientific Workflow Design for Mere Mortals*, Future Generation Computer Systems, 25 (2009), pp. 541–551.

[76] A. A. MEDEIROS, A. WEIJTERS, AND W. M. P. VAN DER AALST, *Using genetic algorithms to mine process models: Representation, operators and results*, Beta, Research School for Operations Management and Logistics, 2005.

[77] J. E. MENDOZA, C. GUÉRET, M. HOSKINS, H. LOBIT, V. PILLAC, T. VIDAL, AND D. VIGO, *Vrp-rep: the vehicle routing community repository*, in Third Meeting of the EURO Working Group on Vehicle Routing and Logistics Optimization (VeRoLog). Oslo, Norway, 2014.

[78] R. S. MICHALSKI AND J. WNEK, *Comparing symbolic and subsymbolic learning: Three studies*, Machine Learning: a Multistrategy Approach, 4 (1993).

[79] A. A. Mitsyuk, I. S. Shugurov, A. A. Kalenkova, and W. M. P. van der Aalst, *Generating event logs for high-level process models*, Simulation Modelling Practice and Theory, 74 (2017), pp. 1–16.

[80] T. Murata, *Petri nets: Properties, analysis and applications*, Proceedings of the IEEE, 77 (1989), pp. 541–580.
PM101 - 1.

[81] N. Patki, R. Wedge, and K. Veeramachaneni, *The synthetic data vault*, in Data Science and Advanced Analytics (DSAA), 2016 IEEE International Conference on, IEEE, 2016, pp. 399–410.

[82] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, *Scikit-learn: Machine learning in Python*, Journal of Machine Learning Research, 12 (2011), pp. 2825–2830.

[83] J. d. S. Pedro and J. Cortadella, *Discovering Duplicate Tasks in Transition Systems for the Simplification of Process Models*, in Business Process Management, M. L. Rosa, P. Loos, and O. Pastor, eds., no. 9850 in Lecture Notes in Computer Science, Springer International Publishing, Sept. 2016, pp. 108–124.

[84] K. E. N. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, *A Design Science Research Methodology for Information Systems Research*, Journal of Management Information Systems, 24 (2008), pp. 45–77.

[85] J. R. Quinlan, *C4. 5: programs for machine learning*, Elsevier, 2014.

[86] J. Ribeiro and J. Carmona, *A Method for Assessing Parameter Impact on Control-Flow Discovery Algorithms*, in Transactions on Petri Nets and Other Models of Concurrency XI. Lecture Notes in Computer Science, vol 9930., M. Koutny, D. J., and J. Kleijn, eds., Springer, 2016.

[87] J. Ribeiro, J. Carmona, M. Misir, and M. Sebag, *A Recommender System for Process Discovery*, in Business Process Management, 8659, Springer, 2014, pp. 67–83.

[88] D. Rosca and C. Wild, *Towards a flexible deployment of business rules*, Expert Systems with Applications, 23 (2002), pp. 385–394.

[89] A. Rozinat, A. A. de Medeiros, C. W. Günther, A. Weijters, and W. M. P. van der Aalst, *Towards an evaluation framework for process mining algorithms*, BPM Center Report, 0706 (2007).

[90] A. Rozinat and W. M. P. van der Aalst, *Decision mining in ProM*, in Business Process Management, S. Dustdar, J. L. Fiadeiro, and A. P. Sheth, eds., Vienna, Austria, 2006, Springer.

[91] ——, *Conformance checking of processes based on monitoring real behavior*, Information Systems, 33 (2008), pp. 64–95.

[92] N. Russell, A. H. M. ter Hofstede, W. M. P. van der Aalst, and N. Mulyar, *Workflow controlflow patterns: A revised view*, Tech. Rep. 06-22, 2006.

[93] G. Schimm, *Mining exact models of concurrent workflows*, Computers in Industry, 53 (2004), pp. 265–281.

[94] S. Siegel and N. J. Castellan Jr, *Nonparametric statistics for the behavioral sciences*, Mcgraw-Hill Book Company, New York, 2 ed., 1988.

[95] T. Stocker and R. Accorsi, *Secsy: Security-aware synthesis of process event logs*, in Workshop on Enterprise Modelling and Information Systems Architectures, 2013, pp. 71–84.

[96] N. Tax, X. Lu, N. Sidorova, D. Fahland, and W. M. van der Aalst, *The Imprecisions of Precision Measures in Process Mining*, Information Processing Letters, 135 (2018), pp. 1–8.

[97] E. UCHOA, D. PECIN, A. PESSOA, M. POGGI, T. VIDAL, AND A. SUBRAMANIAN, *New benchmark instances for the capacitated vehicle routing problem*, European Journal of Operational Research, 257 (2017), pp. 845–858.

[98] W. M. P. VAN DER AALST, *The application of Petri nets to workflow management*, Journal of circuits, systems, and computers, 8 (1998), pp. 21–66.
PM101 - 1.

[99] ——, *Process Mining: Data Science in Action*, Springer, 2016.

[100] W. M. P. VAN DER AALST, A. ADRIANSYAH, A. K. A. D. MEDEIROS, F. ARCIERI, T. BAIER, T. BLICKLE, J. C. BOSE, P. V. D. BRAND, R. BRANDTJEN, J. BUIJS, A. BURATTIN, J. CARMONA, M. CASTELLANOS, J. CLAES, J. COOK, N. COSTANTINI, F. CURBERA, E. DAMIANI, M. D. LEONI, P. DELIAS, B. F. V. DONGEN, M. DUMAS, S. DUSTDAR, D. FAHLAND, D. R. FERREIRA, W. GAALOUL, F. VAN GEFFEN, S. GOEL, C. GUNTHER, A. GUZZO, P. HARMON, A. T. HOFSTEDE, J. HOOGLAND, J. E. INGVALDSEN, K. KATO, R. KUHN, A. KUMAR, M. L. ROSA, F. MAGGI, D. MALERBA, R. S. MANS, A. MANUEL, M. MCCREESH, P. MELLO, J. MENDLING, M. MONTALI, H. R. MOTAHARI-NEZHAD, M. ZUR MUEHLEN, J. MUNOZ-GAMA, L. PONTIERI, J. RIBEIRO, A. ROZINAT, H. S. PÉREZ, R. S. PÉREZ, M. SEPULVEDA, J. SINUR, P. SOFFER, M. SONG, A. SPERDUTI, G. STILO, C. STOEL, K. SWENSON, M. TALAMO, W. TAN, C. TURNER, J. VANTHIENEN, G. VARVARESSOS, H. M. W. VERBEEK, M. VERDONK, R. VIGO, J. WANG, B. WEBER, M. WEIDLICH, T. WEIJTERS, L. WEN, M. WESTERGAARD, AND M. WYNN, *Process mining manifesto*, Lecture Notes in Business Information Processing, 99 (2012), pp. 169–194.

[101] W. M. P. VAN DER AALST, A. ADRIANSYAH, AND B. F. VAN DONGEN, *Causal nets: a modeling language tailored towards process discovery*, in International conference on concurrency theory, Springer, 2011, pp. 28–42.

[102] ——, *Replaying history on process models for conformance checking and performance analysis*, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 2 (2012), pp. 182–192.

[103] W. M. P. VAN DER AALST, A. BOLT, AND S. J. VAN ZELST, *RapidProM: Mine Your Processes and Not Just Your Data*, in RapidMiner: Data Mining Use Cases and Business Analytics Applications, Chapman & Hall/CRC Data Mining and Knowledge Discovery Series, Chapman & Hall/CRC, 2 ed., 2018.

[104] W. M. P. VAN DER AALST, J. BUIJS, AND B. F. VAN DONGEN, *Towards improving the representational bias of process mining*, in Data-Driven Process Discovery and Analysis, Springer, 2012, pp. 39–54.
PM101 - 1.

[105] W. M. P. VAN DER AALST AND A. H. TER HOFSTEDE, *Verification of workflow task structures: A petri-net-baset approach*, Information systems, 25 (2000), pp. 43–69.

[106] W. M. P. VAN DER AALST, A. J. M. M. WEIJTERS, AND L. MARUSTER, *Workflow Mining: Discovering Process Models from Event Logs*, IEEE Transactions on Knowledge and Data Engineering, 16 (2004), pp. 1128–1142.

[107] W. M. P. VAN DER AALST, T. WEIJTERS, AND L. MARUSTER, *Workflow mining: Discovering process models from event logs*, Knowledge and Data Engineering, IEEE Transactions on, 16 (2004), pp. 1128–1142.

[108] J. M. E. VAN DERWERF, B. F. VAN DONGEN, C. A. HURKENS, AND A. SEREBRENIK, *Process discovery using integer linear programming*, Fundamenta Informaticae, 94 (2009), pp. 387–412.

[109] B. F. VAN DONGEN, J. CARMONA, AND T. CHATAIN, *A Unified Approach for Measuring Precision and Generalization Based on Anti-Alignments*, 2016.

[110] B. F. van Dongen, A. A. De Medeiros, and L. Wen, *Process mining: Overview and outlook of petri net discovery algorithms*, in Transactions on Petri Nets and Other Models of Concurrency II, Springer, 2009, pp. 225–242.

[111] K. M. van Hee, J. Hidders, G.-J. Houben, J. Paredaens, and P. Thiran, *On the relationship between workflow models and document types*, Information Systems, 34 (2009), pp. 178–208.

[112] K. M. van Hee and Z. Liu, *Generating Benchmarks by Random Stepwise Refinement of Petri Nets.*, in ACSD/Petri Nets Workshops, 2010, pp. 403–417.

[113] K. M. van Hee, N. Sidorova, and J. M. van der Werf, *Business process modeling using petri nets*, in Transactions on Petri Nets and Other Models of Concurrency VII, Springer, 2013, pp. 116–161.

[114] S. J. van Zelst, B. F. van Dongen, W. M. P. van der Aalst, and H. M. W. Verbeek, *Discovering workflow nets using integer linear programming*, Computing, (2017), pp. 1–28.

[115] S. K. vanden Broucke, C. Delvaux, J. Freitas, T. Rogova, J. Vanthienen, and B. Baesens, *Uncovering the relationship between event log characteristics and process discovery techniques*, in Business Process Management Workshops, Springer, 2014, pp. 41–53.

[116] S. K. L. M. vanden Broucke, *Advances in Process Mining*, PhD thesis, Katholieke Universiteit Leuven, Leuven, 2014.

[117] S. K. L. M. vanden Broucke and J. De Weerdt, *Fodina: A robust and flexible heuristic process discovery technique*, Decision Support Systems, 100 (2017), pp. 109–118.

[118] S. K. L. M. vanden Broucke, J. De Weerdt, B. Vanthienen, Jan, and B. Baesens, *Determining process model precision and generalization with weighted artificial negative events*, Knowledge and Data Engineering, IEEE Transactions on, 26 (2014), pp. 1877–1889.

[119] J. VENABLE, J. PRIES-HEJE, AND R. BASKERVILLE, *FEDS: a framework for evaluation in design science research*, European Journal of Information Systems, 25 (2016), pp. 77–89.

[120] H. M. W. VERBEEK, J. C. A. M. BUIJS, B. F. VAN DONGEN, AND W. M. P. VAN DER AALST, *Xes, xesame, and prom 6*, in Information Systems Evolution, P. Soffer and E. Proper, eds., Springer, 2011, pp. 60–75.

[121] J. VOM BROCKE AND M. ROSEMANN, *Handbook on business process management 1*, Springer, 2010.

[122] D. VOSS AND OTHERS, *Design and Analysis of Experiments*, New York: Springer, 1999.

[123] B. VÁZQUEZ-BARREIROS, M. MUCIENTES, AND M. LAMA, *Mining Duplicate Tasks from Discovered Processes*, in Algorithms and Theories for the Analysis of Event Data (ATAED 2015), 2015.

[124] ——, *ProDiGen: Mining complete, precise and minimal structure process models with a genetic algorithm*, Information Sciences, 294 (2015), pp. 315–333.

[125] J. WANG, R. K. WONG, J. DING, Q. GUO, AND L. WEN, *Efficient Selection of Process Mining Algorithms*, Services Computing, IEEE Transactions on, 6 (2013), pp. 484–496.

[126] P. WEBER, B. BORDBAR, AND P. TINO, *A Framework for the Analysis of Process Mining Algorithms*, IEEE Transactions on Systems, Man, and Cybernetics: Systems, 43 (2013), pp. 303–317.

[127] A. WEIJTERS AND J. RIBEIRO, *Flexible Heuristics Miner (FHM)*, in 2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM), Apr. 2011, pp. 310–317.

[128] A. WEIJTERS, W. M. P. VAN DER AALST, AND A. A. DE MEDEIROS, *Process mining with the heuristics miner-algorithm*, Technische Universiteit Eindhoven, Tech. Rep. WP, 166 (2006).

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

[130] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*, Morgan Kaufmann, 2016.

[131] D. H. Wolpert and W. G. Macready, *No free lunch theorems for optimization*, IEEE transactions on evolutionary computation, 1 (1997), pp. 67–82.

[132] M. Zur Muehlen and J. Recker, *How much language is enough? Theoretical and practical use of the business process modeling notation*, in Seminal Contributions to Information Systems Engineering, Springer, 2013, pp. 429–443.