Use Exceptional Model Mining in Process Mining to discover interesting cases

Frints, Noud J.A.

Award date:
2021

Use Exceptional Model Mining in Process Mining to discover interesting cases

Master of Science Thesis

For obtaining the degree of Master of Science in Computer Science and Engineering at the department of Mathematics and Computer Scienceof Eindhoven University of Technology

Noud Frints - 0909421
noudfrints@gmail.com

March 19, 2021

Supervisors:

Wouter Duivesteijn	Mathematics and Computer Science
Dirk Fahland	Mathematics and Computer Science
Murat Firat	Industrial Engineering and Innovation Sciences

Acknowledgments

I would like to thank Wouter Duivesteijn and Dirk Fahland for providing guidance and feedback during my project, as well as proof reading my thesis while providing valuable feedback.
Next, I would like to thank Wouter Duivesteijn, Dirk Fahland and Murat Firat for being on my graduation committee.
Finally, a special word of thanks to my partner, for coping with me and allowing me to work on this project while facing each other at the dinner table, due to Covid-19 measures.

Abstract

When looking at a business process, one can obtain various insights and information by analyzing the event logs that the business has created when executing this process. In an ideal world, everything would go as desired and designed by the process manager, but more often than not it happens that deviations in the process occur due to inconsistencies in the process or due to the deviating and uncontrollable nature of the process being logged (e.g., an emergency in a hospital). Process Mining tries to reduce the number of deviations by extracting models from this event log by creating valuable subsets and gaining insightful observations from these models. However, since the logs are so diverse, it is not easy to obtain these subsets. It is often more easy to indicate which attributes or factors are of importance when analyzing a process, e.g., long duration of cases is always bad practice. That is where Exceptional Model Mining (EMM) could be of use. Exceptional Model Mining is a supervised data mining framework which can create subgroups based on multiple target attributes for which certain descriptions are applicable. These obtained subgroups might prove to be an interesting starting point when analyzing the business process.

Key Words: Exceptional Model Mining, Process Mining, data mining, process discovery, model, clustering, patterns, groups, exceptions

Contents

1 Introduction

1.1 Background and Motivation

Everywhere around us are processes. Some of them are obvious and well structured, e.g., placing an order on Amazon. Others are pretty well hidden and not immediately noticeable, e.g., the play style of a football team based on the tactics the trainer developed to perform better than the opposing team. In order for these services and activities to operate smoothly, the underlying process has to be functioning and operating as planned. Most modern day businesses store all events occurring. These events and their attributes are all stored in logs. Process Mining [20] tries to obtain insights, information, and relations between events from the business *event logs*, to discover the underlying process and remove bottlenecks. This is all done to improve the execution of the process. Due to the diverse and unpredictable nature of most real-life processes, event logs store many cases with various, very varying event flows, or process executions, which makes the discovery of the underlying process a very difficult and tedious task.

1.2 Problem Identification

In many applications, stakeholders prefer to have only a subset of the cases that contain deviating behaviour analysed in order to know more about them. Examples of interesting subsets include:

- Cases with high or low performance.
- Cases with high profits for the company.
- Unfinished or canceled cases.
- Cases that resulted in user complaints.

When discovering and analysing these subsets of cases, shared attributes (attributes that are the same for all cases in the subset) are essential. For example, discovering cases with low performance that were all executed by the same department might be an indication of an issue within that department.

Some techniques that are able to discover similarity between cases have already been developed. Trace clustering [23] groups cases in these event logs together, based on their similarities in behaviour, effectively reducing the number of control flow variations. This is a desirable technique, but it still requires a lot of work to discover which clusters show behaviour that is undesired, based on a user specified attribute. Anomaly detection [13] allows for the detection of cases that show behaviour that does not match with the desired business process. This requires prior process knowledge and still does not allow for the discovery of subsets, based on a user specified attribute.

The discovery of shared attributes, or relations, between items is a classic data mining task. When discovering relations between items with respect to a property of interest, or a *target attribute*, subgroup discovery [14] comes to mind. Subgroup discovery allows for the discovery of rules describing subsets of the dataset that are sufficiently large and contain diverging behaviour, when compared to one target attribute. However, since most processes are very complex and depend on several factors that influence the event sequence, only one target attribute will almost never be sufficient. For example, a process stakeholder is not satisfied with the execution time of some cases. To address this issue, one should not only consider the total case execution time, but also the number of events that were executed in that case. All events take time to complete, thus increasing the total execution time. To be able to discover cases that have a deviating relation between the total execution time and the number of events, at least these two attributes should be considered. If the need for two target attributes already exists in this simple example, more complex interests will require more target attributes. This is not only the case for Process Mining, in other datasets, the discovery of interesting relations based on multiple attributes is also a desired ability. That is why Exceptional Model Mining [8] was introduced. Exceptional Model Mining allows for the discovery of interesting subgroups based on a more complex target model, based on one or more target attributes. In this sense, Exceptional Model Mining can be seen as a generalization of subgroup discovery.

1.3 Research Questions

When researching the possibilities of using Exceptional Model Mining for Process Mining, we answer the main question,

Can Exceptional Model Mining be used in Process Mining to discover subsets of cases that contain deviating behaviour based on user specified attributes?

to determine whether it is possible to use Exceptional Model Mining for Process Mining to discover deviating cases, and how to use the obtained results. We find an answer to this question by answering the following sub questions.

- *How can we obtain traces that contain deviating behaviour from event logs using Exceptional Model Mining?*
  The event logs contain traces which all consist of events. These traces and events all have attributes, which are used to describe them. Since these traces are stored as a sequence of events, stored in multiple rows of a table, traditional, flat-table data mining techniques cannot be applied for trace analysis. Thus, we need to transform the event log.

- *How should we transform the event logs?*
  To be able to apply Exceptional Model Mining to the event log, the event log should be transformed to a case-attributes format, where each row of the dataset contains all attributes of a trace and its events in different cells.

All information of a single trace is now stored in a single row. This way it has the same format as a classic, flat-table dataset, which is compatible with Exceptional Model Mining.

- *Which information should these transformed traces contain?*
  Traces contain a lot of static and dynamic attributes. Is it enough to only include static attributes, e.g., the frequency of the events within a trace, or the existence of a loop? Or should dynamic attributes, e.g., where the loop exists also be listed? And if so, how should they be described?

- *How should the results obtained by Exceptional Model Mining be analysed to obtain useful insights?*
  Exceptional Model Mining will return subgroups, but what should we do with these? Are the descriptive attributes an indication of where a problem resides? Or do the subgroups tell us something else? And if so, what has to be done, such that we do get an indication where the problem resides?

- *How can these interpreted results help us with Process Mining?*
  The obtained subgroups will have attributes to describe them, and differ in distribution for the target attributes, when compared to the dataset. But how should we then use these subgroups to start with Process Mining? Can we just use the description to filter on the traces that contain these descriptive attributes, or is more processing needed?

To evaluate the possibility of using Exceptional Model Mining for real life businesses, we provide a case study where we apply our method on a compensation handling process [2]. The main contribution of this thesis is that we apply Exceptional Model Mining techniques in the context of the Process Mining domain, to discover the most interesting patterns in a subset of cases based on multiple target attributes. These subsets of cases can then be further investigated using more traditional Process Mining techniques to see whether they indeed show different behaviour than the desired behaviour present in the majority of the cases.

1.4 Research Method

When the idea of using Exceptional Model Mining for Process Mining to discover subgroups was born, we started with a literature study in both the *Data Mining* and *Process Mining* research fields, to determine whether something similar was already possible. The findings of this study can be found in Section 3.

When performing this literature study, we learned that the existing Process Mining techniques all require a fair amount of base process knowledge. This knowledge is used to discover which subsets of traces contain the desired behaviour, and which subsets of traces contain deviating behaviour. Obtaining

this knowledge can be a tedious task, since it either requires communication with the process owners, or the selection of traces that occur most often. However, both options only give the user information about the desired execution of the process, thus the user still does not know anything about possible deviations, e.g., bottlenecks. Exceptional Model Mining is able to discover subsets of traces that show deviating behaviour, which is an indication on where possible bottlenecks are located. This allows for the absence of base knowledge, thus speeding up the bottleneck discovery process. We also learned that there would be a mismatch in the data formats. Exceptional Model Mining operates on flat-table style data formats, where every row describes an item in the dataset, with the corresponding attributes stored in the columns, whereas Process Mining operates on the XES format, where every trace contains a sequence of events, the trace attributes, and event attributes. So the need for data transformation arose. More information on this topic are found in Section 4.

We decided to create a transformation tool, that is able to transform the event log into a flat-table like format, where every row represents a trace. The row identifier is used as *CaseID*, the event attributes are counted and these counts are stored, and since the trace attributes are already at trace level, they can be stored as they are. This way the sequence of activities and the event attribute-value pairs are lost, but this can be recovered from the original event log. This is elaborated in Section 5. We also created our own Exceptional Model Mining framework implementation, because this would make it easier to add extra functionality, such as extra quality measures, and fix issues that arose during the discovery process, such as mismatches in data and variable types.

We used a publicly available event log describing a simple process. We increased the number of cases, and manually modified a subset of the cases. This synthetic data allows us to have a ground truth when evaluating our results when exploring the possibilities of Exceptional Model Mining for Process Mining. The event log used [2], describes the simple process of evaluating compensation requests. Requests are registered and reviewed, after which they are accepted, denied, or reevaluated. The event log only contains 6 cases, which does not contain enough information to be able to really validate any obtained results. Because of this we decided to extend the event log by discovering the process tree and generating a user-specified number of traces from this process tree using *PM4PY* [5]. The traces are adapted such that they contain all attributes and have similar duration for every event. This way, the adapted event log resembles the original event log as close as possible. Finally we altered the event log in two ways, such that it contains a subset of traces that contain deviating performance behaviour. More information can be found in Section 6, up to and including Section 6.1. We decided to use a relatively simple event log, because this would allow us to perform our own alterations, which, if discovered are then known to be correct, instead of using a real-life dataset, where findings would have to be communicated with the process stakeholders. The application on real-life event logs is something that can still be done in the future.

Finally, we used our Exceptional Model Mining framework implementation to discover subgroups using three different quality measures, which are explained in Section 6.1.1. The discovered subgroups are validated using our own analysis. After this verification, we used the information found in the subgroup to filter the original event log, such that it only contains the traces with the deviating behaviour. Using the Inductive Miner, we generated process models, which we analysed to see whether they show the same results as we obtained using Exceptional Model Mining.

1.5 Contribution and Results

Using Exceptional Model Mining on a transformed event log, we were able to discover subgroups of traces that contain deviating performance behaviour. The attributes of the traces in these subgroups were analysed, to find attributes that can be used to filter the original event log. The resulting event logs were analysed by generating their corresponding process models. These process models also showed us that the corresponding subgroup indeed contained traces which deviate from the rest.

We also applied more traditional Process Mining techniques to the event log, to check whether these were able to discover the subset of traces with deviating behaviour. This verification was performed using as little process knowledge as possible, as this is something that is not required when applying Exceptional Model Mining on a transformed event log. With the limited resources available, we were unable to discover the same subset of cases using these more traditional Process Mining techniques.

1.6 Overview

In **Chapter 2** we will briefly introduce why Exceptional Model Mining is useful for Process Mining. Secondly Exceptional Model Mining will be introduced. In this section we will describe how Exceptional Model Mining works and which measures it uses to achieve its goal. Finally, Process Mining is introduced. In this section we will describe how Process Mining works and some of the measures it uses to achieve its goal.

Chapter 3 first will contain some data mining techniques similar to Exceptional Model Mining. Secondly it will contain some techniques used in Process Mining. Finally it will show some Process Mining techniques which are in some way related to the use of Exceptional Model Mining for Process Mining.

In **Chapter 4** we will elaborate further why Exceptional Model Mining could be useful for Process Mining. We will do so by discussing what we want to achieve and how we want to achieve this.

Chapter 5 will elaborate on how Event Data should be altered to be useful for Exceptional Model Mining.

in **Chapter 6** the experimental results obtained will be reviewed and we will discuss whether these results are viable or not.

2 Motivation and Preliminaries

2.1 Motivation

The main tasks of Process Mining are to obtain a model from the available Event Data, which can be used to find bottlenecks in the process that is logged in the Event Data, and find deviations from the happy flow (the optimal execution of the process). These bottlenecks and deviations are then to be resolved and removed to get as close to this happy flow as possible. When trying to obtain a process model to discover the bottlenecks and deviations, a fair amount of user interaction and process knowledge is required. This user action is required because subsets of cases that behave differently from each other should be separated, to be able to distinguish good cases from the bad ones, which contain bottlenecks or deviations [9].

Since, usually, a process' log contains many different traces, knowing where to start with applying filters can be quite difficult. That is where Exceptional Model Mining comes to play. Exceptional Model Mining can mine subgroups from datasets which contain data that deviates from the rest of the dataset, based on a user induced quality measure. If Exceptional Model Mining is capable of discovering subgroups of traces in a process, which contain data that differs from the norm, these subgroups might contain bottlenecks and give the user an indication on where to start looking for such bottlenecks.

Sections 2.2 and 2.3 will provide the reader with the required preliminary knowledge on the topics of Exceptional Model Mining and Process Mining respectively.

2.2 Exceptional Model Mining

What is Exceptional Model Mining?[1]
To be able to answer this question, the following questions will be answered.

- What is the goal of Exceptional Model Mining?
- How does it achieve this?
- Which strategies does it use?
- Which possibilities are there for quality measures?

2.2.1 What is the goal of Exceptional Model Mining?

Data mining is the process of discovering patterns in large data sets involving methods at the intersection of machine learning, statistics, and database systems[29]. Finding interesting subsets of data that show divergent behaviour is a

[1] The information used in this Section is all obtained from material by Wouter Duivesteijn et al. [6][8]

classic data mining task. Subgroup discovery (SD)[14] is an instance of data mining that tries to find such interesting subsets based on one single target attribute. These subsets can be found by comparing the distribution of this attribute in such a subgroup and in the entire dataset and checking whether these distributions differ, it is also possible to compare the distribution of the subset against the complement of the subgroup. A limitation of SD is that the assessment of a subgroup's exceptionality can be based on only *one* single attribute having a deviating distribution. To tackle this, Exceptional Model Mining (EMM) was introduced. EMM is a local, supervised pattern mining framework that allows its users to find subgroups that show divergent behaviour based on multiple attributes. When the target concept in a dataset can no longer be described using a single attribute, but we still want to find interesting subgroups, EMM will offer a solution. EMM splits the dataset at hand in two subsets, the descriptive and target attributes. The descriptive attributes are used to define the subgroups, and the target attributes are used to evaluate the created subgroups using a user-specified quality measure.

Any symbols used in this Section may pop up at any moment. From this point on we assume the meaning is understood.
Assume a dataset Ω to be a bag of N *records* r $\in \Omega$ of the form

$$r = (a_1, ..., a_k, l_1, ...l_m)$$

where k and m are positive integers. $a_1, ...a_k$ are called the *descriptive attributes* or *descriptors* of r and $l_1, ...l_m$ are called the *target attributes* or *targets* of r. The descriptors come from an unrestricted domain $\mathcal{A}$. In Section 2.2.4 we will learn models from a selected *model class* over the targets.

For the definition of subgroups *descriptions* need to be defined. In practice, descriptions will usually be taken from a description language $\mathcal{D}$, which is chosen by the user. What these subgroups exactly look like, will be discussed later. Mathematically, we will define descriptions as functions $D : \mathcal{A} \rightarrow \{0, 1\}$. A description D *covers* a record r^i iff $D(a_1^i, ..a_k^i) = 1$. Informally, a description D can be seen as the conjunction of attributes taken from description language $\mathcal{D}$.

Definition (Subgroups)**.** A *subgroup* corresponding to a description D is the bag of records $G_D \subseteq \Omega$ that D covers, i.e.

$$G_D = \{r^i \in \Omega | D(a_1^i, ..a_k^i) = 1\}$$

If no confusion can arise, we can omit D, thus referring to a subgroup as G. Subgroups and their definitions can freely be associated with each other. When we have a particular subgroup G in mind, we will use n to denote the number of items in that subgroup: $n = |G|$, which is also called the *coverage* of the description. The complement of the subgroup is denoted by G^c, and we use n^c for its number of items. Hence $G^c = \Omega \backslash G$ and $n^c = N - n$.

To be able to evaluate a candidate subgroup, the need for *quality measures* arises. For every description D in $\mathcal{D}$, a quality function indicates how much the subgroup G_D deviates from the norm.

Definition (Quality Measure)**.** A *quality measure* is a function $\varphi : D \to \mathbb{R}$ that assigns a numeric value to a description D.

Since subgroups are defined by descriptions, $\varphi(G)$ is referring to the quality of a subgroup

2.2.2 How does it achieve this goal?

EMM is a data mining framework that can be described as a generalization of the Subgroup Discovery framework. SD's and EMM's goal is to find descriptions which satisfy certain user-specified constraints. These descriptions usually include lower bounds for the subgroup size ($|G_D| \geq lb_1$) and quality of the description ($\varphi(D) \geq lb_2$). More constraints may be introduced if required by the question at hand. Thus, e.g., domain experts may for instance, request an upper bound on description complexity.

The most common SD algorithms traverse the search space of candidate descriptions in a general-to-specific way: They treat the space as a lattice whose structure is defined by a *refinement operator* $\eta : \mathcal{D} \to 2^{\mathcal{D}}$. This operator determines how the descriptions can be extended with atomic additions to become more complex. Most applications assume η to be a *specialization operator*: every description D_i that is an element of the set $\eta(D_j)$, is a more specialized description compared to D_j itself. Thus, when a description is more specialized, it means that the description has more descriptive attributes, which describes a smaller, more specialized subgroup. The algorithm results in a ranked list of descriptions that satisfy the user-defined constraints.

As already mentioned, EMM can be seen as a generalization of of SD. Rather than one single target variable, EMM uses a more complex target complex. An instance of EMM is defined by the combination of a *model class* over the targets and a *quality measure* over this model class. When an instance has been defined, subgroups are generated to be evaluated. A model on the targets is then induced for each subgroup under consideration. This model is learned from the data belonging to the subgroup only. Next, the quality measure is used to evaluate the subgroups to determine which subgroups are the most interesting. The typical quality measure in EMM indicates how exceptional the model based on the targets in the subgroup is. Most of the time, it is either compared to the models based on the targets in the entire dataset or to the models based on the targets in the complement of the dataset.

The goal of EMM is to find interesting subgroups in a dataset.The interestingness is concretely defined by the end user. Nevertheless, to provide a more

precise description on what we find interesting, we provide the following task definition [7]

Problem Statement (Top-q Exceptional Model Mining)**.** *Given a dataset* Ω*, description language* $\mathcal{D}$*, quality measure* φ*, positive integer* q*, and set of constraints* $\mathcal{C}$*. The* Top-q *Exceptional Model Mining task delivers the list* $\{D_1, ..., D_q\}$ *of descriptions in the language* $\mathcal{D}$ *such that*

- $\forall_{1 \leq i \leq q} : D_i$ satisfies all constraints in $\mathcal{C}$;
- $\forall_{i,j} : i < j \Rightarrow \varphi(D_i) \geq \varphi(D_j)$;
- $\forall_{D \in \mathcal{D} \setminus \{D_1,...,D_q\}}$: D satisfies all constraints in $\mathcal{C} \Rightarrow \varphi(D) \leq \varphi(D_q)$

Informally, we find the q best-scoring descriptions from description language $\mathcal{D}$ that satisfy all constraints in $\mathcal{C}$. This set contains both the user-induced constraints and search strategy limitations. These limitations include information about the exact choice we make for the refinement operator η, which defines the generation of candidate subgroups out of existing subgroups, and the limits for exploring the search space.

2.2.3 Which search strategies does it use?

Finding interesting subsets of the data is the goal of EMM. This means that the search space can be exponentially large. Using brute force thus is no possibility, so the need for more sophisticated search strategies arises. Two trains of thought are currently running in the community on how to tackle this problem. The one following canonical SD papers restricts the attributes in the dataset to be nominal and imposes an anti-monotonicity constraint on the used quality measure. The other resorts to heuristic search. This allows the attributes to be numeric as well and facilitates a generic quality measure. EMM is developed to capture any notion of interestingness in subgroups, thus allowing for any quality measure and numeric attributes are more important. Therefore, the heuristic path is chosen.

Usually the *beam search* strategy is chosen, which performs a level-wise search. On each level, the best w (for *search width*) descriptions according to our quality measure φ are selected and refined to create the candidate descriptions for the next level. This strategy is constrained by an upper bound on the complexity of the description (*search depth*, d) and a lower bound on the support of the corresponding subgroup. Informally, this means that the upper bound is the number of logical conjunctions of single attributes. The lower bound is the number of elements in the corresponding subgroup. This strategy combines both the advantages of a greedy method with those of the implicit parallel search, since only the best w possibilities are considered. Due to this it is less likely to end up in a local optimum when compared to a greedy approach. While selecting the best w best descriptions at each level keeps the process focused, hence tractable.

An important part of the beam search strategy is generating a set of candidate descriptions for the next level, by refining another description. This process is guided by the refinement operator η and the description language $\mathcal{D}$. Our description language $\mathcal{D}$ consists of logical conjunctions of conditions on single attributes.

2.2.4 What possibilities are there for Quality Measures?

As stated before, an EMM instance is defined by the choice of model class over the targets and a quality measure over the model class. Having chosen a model class, we need to define a quality measure that extracts characteristics from the learned models and extracts from these models a quantification on how different these models are from each other. Usually that quantification is pretty straightforward to design. E.g., if the model class is a regression model with two variables, one could take the difference in slope for each model as a quality measure. This quantification would usually not be enough to design a proper quality measure. Very small subsets can deviate from the norm, but usually are not very noteworthy because of their small size. Therefore, we usually want to include the size of a subgroup in our quality measure.

In some of the canonical quality measures for Subgroup Discovery, such as *Weighted Relative Accuracy (WRAcc)*[17], the size of a subgroup is directly represented by a factor n or $\sqrt{n}$. Even though this simplicity is appealing, one might argue that it is counter-intuitive to have a factor in a quality measure that explicitly favors subgroups covering the entire dataset over smaller subgroups. A more sophisticated way to represent the subgroup size is to multiply the quantification of model difference with the *entropy* of the split between the subgroup and its complement. The entropy captures the information content of such a split and favours balanced splits (1 bit of information for a 50/50 split) over skewed splits (0 bits of information for the extreme case of either subgroup or complement being empty). In this case the entropy function is defined as

$$\varphi_{ef}(D) = -\frac{n}{N}log(\frac{n}{N}) - \frac{n^c}{N}log(\frac{n^c}{N})$$

2.3 Process Mining

What is Process Mining?
To be able to answer this question, we will answer the following subquestions:

- What does Process Mining try to achieve?
- What is an event log?
- What is Event Data and how to obtain an event log?
- What is a Process Model?
- How can we obtain a Process Model?

- Which quality criteria should be taken into account?

2.3.1 What does Process Mining try to achieve?

Process Mining aims to extract novel insights from Event Data. Process Discovery plays an important role in Process Mining. The goal is to discover a process model that represents the event sequences found in the event data in terms of start-to-end behaviour. These models help understand the behaviour of the process as it actually happens, instead of how the process designer wanted it to be. This helps to identify, remove, predict, and prevent bottlenecks, inefficiencies, and costly variations.

2.3.2 What is an event log?

Events are listed together with their attributes in an event log. Typical attributes usually are Case ID, Activity, the timestamp and other attributes recorded by the logging system. The event log thus represents one or more cases of the process. An example event log can be seen in Figure 1. In this event log, we have CaseID, Activity, Resource, Time and Costs attributes per trace. Each trace has a unique case identifier to distinguish different cases from each other. In this event log, the cases are sorted based on their CaseID, which makes the understanding of the underlying process easier. Another possibility would be to sort them based on their timestamp, which would show the order of events stored in the system. The event log in Figure 1 describes the process of evaluating a compensation request, which happens roughly as follows:

1. A request is registered
2. The request is examined and the corresponding ticket is checked
3. One of the following decisions is made:
    - The request is approved and a compensation is paid
    - The request is denied and the compensation doesn't get paid
    - The request is re-evaluated from step 2 and onward.

The process model we can generate from this log can be seen in Figure 2. How this is done, is discussed in Section 2.3.5.

2.3.3 What is Event Data and how to obtain event logs?

Event Data consists of sequences of events which all lie in the universe of all **events** E, such that for every event $e : e \in E$. These events contain attributes, which all have names. These names reside in the universe of **attribute names** AN, name $n \in AN$. These attributes that are used to describe the event have a certain value, which reside in the universe of **value** V, where value $v \in V$.

	CaseID	Activity	Resource	Time	Costs
0	1	register request	Pete	2010-12-30 10:02:00+00:00	50
1	1	examine thoroughly	Sue	2010-12-31 09:06:00+00:00	400
2	1	check ticket	Mike	2011-01-05 14:12:00+00:00	100
3	1	decide	Sara	2011-01-06 10:18:00+00:00	200
4	1	reject request	Pete	2011-01-07 13:24:00+00:00	200
5	2	register request	Mike	2010-12-30 10:32:00+00:00	50
6	2	check ticket	Mike	2010-12-30 11:12:00+00:00	100
7	2	examine casually	Sean	2010-12-30 13:16:00+00:00	400
8	2	decide	Sara	2011-01-05 10:22:00+00:00	200
9	2	pay compensation	Ellen	2011-01-08 11:05:00+00:00	200
10	3	register request	Pete	2010-12-30 13:32:00+00:00	50
11	3	examine casually	Mike	2010-12-30 14:06:00+00:00	400
12	3	check ticket	Ellen	2010-12-30 15:34:00+00:00	100
13	3	decide	Sara	2011-01-06 08:18:00+00:00	200
14	3	reinitiate request	Sara	2011-01-06 11:18:00+00:00	200
15	3	examine thoroughly	Sean	2011-01-06 12:06:00+00:00	400
16	3	check ticket	Pete	2011-01-08 10:43:00+00:00	100
17	3	decide	Sara	2011-01-09 08:55:00+00:00	200
18	3	pay compensation	Ellen	2011-01-15 09:45:00+00:00	200
19	4	register request	Pete	2011-01-06 14:02:00+00:00	50
20	4	check ticket	Mike	2011-01-07 11:06:00+00:00	100
21	4	examine thoroughly	Sean	2011-01-08 13:43:00+00:00	400
22	4	decide	Sara	2011-01-09 11:02:00+00:00	200
23	4	reject request	Ellen	2011-01-12 14:44:00+00:00	200
24	5	register request	Ellen	2011-01-06 08:02:00+00:00	50
25	5	examine casually	Mike	2011-01-07 09:16:00+00:00	400
26	5	check ticket	Pete	2011-01-08 10:22:00+00:00	100
27	5	decide	Sara	2011-01-10 12:28:00+00:00	200
28	5	reinitiate request	Sara	2011-01-11 15:18:00+00:00	200
29	5	check ticket	Ellen	2011-01-14 13:33:00+00:00	100
30	5	examine casually	Mike	2011-01-16 14:50:00+00:00	400
31	5	decide	Sara	2011-01-19 10:18:00+00:00	200
32	5	reinitiate request	Sara	2011-01-20 11:48:00+00:00	200
33	5	examine casually	Sue	2011-01-21 08:06:00+00:00	400
34	5	check ticket	Pete	2011-01-21 10:34:00+00:00	100
35	5	decide	Sara	2011-01-23 12:12:00+00:00	200
36	5	reject request	Mike	2011-01-24 13:56:00+00:00	200
37	6	register request	Mike	2011-01-06 14:02:00+00:00	50
38	6	examine casually	Ellen	2011-01-06 15:06:00+00:00	400
39	6	check ticket	Mike	2011-01-07 15:22:00+00:00	100
40	6	decide	Sara	2011-01-07 15:52:00+00:00	200
41	6	pay compensation	Mike	2011-01-16 10:47:00+00:00	200

Figure 1: An example event log

To be able to assign a value to an event attribute, we need an Attribute-Value function

$$\# : E \times AN \to V \cup \{\perp\}$$

An example of such an assignment is as follows; "Attribute n of event e has value v"

$$\#_n(e) = v$$

"Attribute n of event e is undefined/has no value"

$$\#_n(e) = \perp$$

These events all have a case identifier, which lies in the universe of all cases, $c \in C$. Cases can have attributes as well, which can be assigned using the attribute-value function:

$$\# : C \times AN \to V \cup \{\perp\}$$

The default attribute of a case is the trace, which is the sequence of events

$$\#_{trace}(c) = \langle e_1, e_2, ...e_n \rangle \in E^*$$

All events in a trace are *ordered in time*, such that

$$\forall_{i,j} 1 \leq i < j \leq n : \#_{time}(e_i) \leq \#_{time}(e_j)$$

holds, also, some event attribute $cid \in AN$ is the case identifier attribute, which is the same for all events in a trace.

$$\exists_{cid} cid \in AN : \forall_i 1 \leq i \leq n : \#_{cid}(e_i) = c$$

If we take a subset of this set of cases we obtain a **sequential log** $L \subseteq C$,

$$\forall_{c_1,c_2} c_1 \neq c_2 \in L : set(\#_{trace}(c_1)) \cap set(\#_{trace}(c_2)) = \emptyset$$

"no two cases share an event/each event belongs to 1 case only." where $set(\langle e_1, e_2, ..., e_n \rangle) = \{e_1, e_2, ..., e_n\}$ is the set of events in a sequence.

A **simple log** $L \subseteq \mathbb{B}(\Sigma^*)$ is a multiset of sequences over activities Σ. It is possible to obtain a **simple trace** of case c such that

$$\underline{c} := \#_{activity}(\#_{trace(c)})$$

which returns a sequence of activity classifier values. An example of a simple trace is $\langle$*register request, examine casually, check ticket, decide, pay compensation*$\rangle$.
A **simple log** can then be obtained such that

$$\underline{L} = [\underline{c} | c \in L]$$

which is a multiset of simple traces.

We can obtain a *simple trace* c_x using the method described above for all six cases in Figure 1 (CaseID) by projecting each event on its activity attribute. This results in Simple Log L_1, containing the following traces:

- c_1: $\langle$*register request, examine thoroughly, check ticket, decide, reject request*$\rangle$
- c_2: $\langle$*register request, check ticket, examine casually, decide, pay compensation*$\rangle$
- c_3: $\langle$*register request, examine casually, check ticket, decide, reinitiate request, examine thoroughly, check ticket, decide, pay compensation*$\rangle$
- c_4: $\langle$*register request, check ticket, examine thoroughly, decide, reject request*$\rangle$
- c_5: $\langle$*register request, examine casually, check ticket, decide, reinitiate request, check ticket, examine casually, decide, reinitiate request, examine casually, check ticket, decide, reject request* $\rangle$
- c_6: $\langle$*register request, examine casually, check ticket, decide, pay compensation*$\rangle$

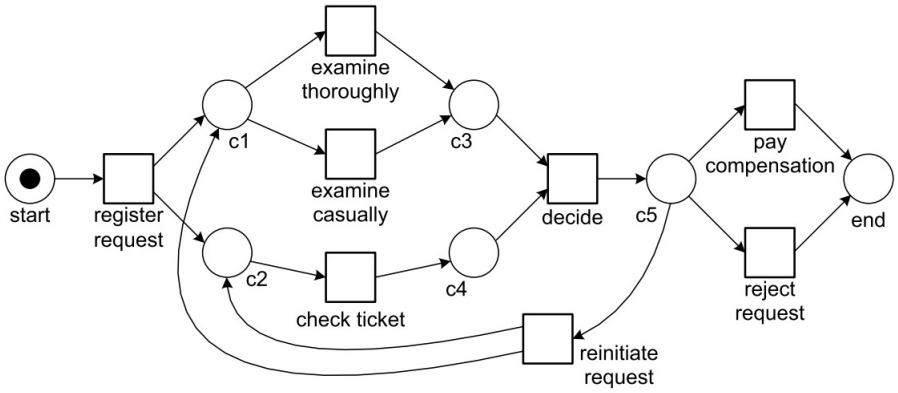


Figure 2: The model obtained by applying Process Discovery techniques to Simple Log L_1

These simple logs are then used in Process Discovery to obtain a model M with $Behaviour(M) \approx L$. These logs contains traces, but since the process it describes is executed often, many traces will have the same sequence. Each unique sequence $\sigma \in L$ is called a trace variant. The more trace variants the log contains, the more complex model M becomes.

From L_1 we can obtain model M_1 using Process Discovery techniques, which can be seen in Figure 2. Some of these techniques are explained in Sections 2.3.5 and 3.2.

2.3.4 What is a Process Model?[2]

The model shown in Figure 2 is also known as a **Petri Net**. Petri Nets are the oldest and best investigated process modeling language describing concurrency. Even though its notation is simple and intuitive, Petri Nets have precise operational semantics and many analysis techniques can be used to analyse them.

Figure 3 shows the Petri Net again with the various constructs highlighted. A Petri Net is a bipartite graph consisting of places and transitions. The network structure is static, but, governed by the firing rule, tokens can flow through the network. The state of a Petri Net is determined by the distribution of tokens over places and is referred to as its *marking*. In the initial marking shown in Fig.3, there is only one token; *start* is the only marked place.

Informally, the *firing rule* describes the behaviour of the Petri Net. In an un-

[2]The information used in this part is obtained from the book Process Mining by Will van der Aalst [1]

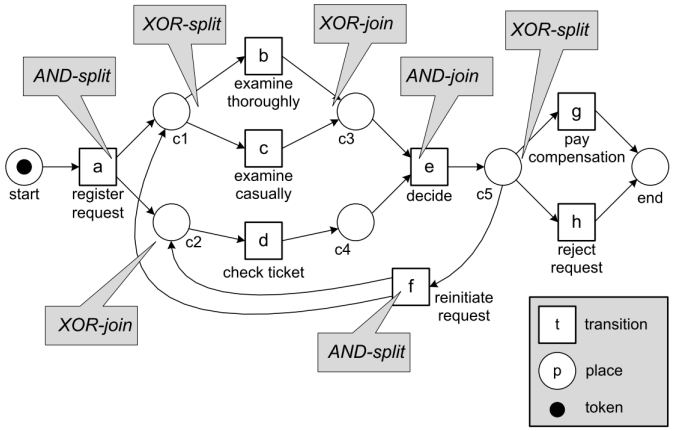


Figure 3: A marked Petri Net

weighted Petri Net (all arcs have multiplicity 1), a transition in the net is able to *fire*, if in all its ingoing places at least one token is available. After firing this transition, one token is produced in all of the transitions' outgoing places. In the model in Figure 3, only transition *register request* can fire, since it contains a token in the *start* place. Firing this transition would produce a token in places c_1 and c_2.

Definition. (Petri Net) A Petri Net is a triplet N = (P,T,F) where P is a finite set of places, T is a finite set of transitions such that $P \cap T = \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs, called the flow relation. A marked Petri Net is a pair (N, M), where $N = (P, T, F)$ is a Petri Net and where $M \in B(P)$ is a multiset over P denoting the marking of the net. The set of all marked Petri Nets is denoted $\mathscr{N}$.

When we have obtained a process model, we can analyse the *behaviour* of that model. The behaviour of a process model can be seen as the set of traces that are allowed by that model. If the model contains a loop, the size of this set can be infinitely large, as every iteration of that loop results in a new trace variant.

2.3.5 How can we obtain a Process Model?

Process Discovery is the task of transforming an event log to a Process Model. To obtain a valid model, the event log should contain enough trace variants, such that the obtained model is likely to cover the entire process written in the event log. The fact that some deviating traces also reside in the event log should also be taken into account. This means that trace variants that only occur once or twice are likely to be an incorrect execution of the process, and should not be taken into account when designing a model.

Different strategies, or algorithms, have been developed to discover a process from an event log. One of these algorithms is the **α-algorithm**. This algorithm takes a log as input and constructs a model. For each pair of events (e_i, e_j) it tries to discover their ordering relation. These ordering relations are defined as follows:

- Succession: $e_i \succ e_j$ iff e_i precedes e_j in some trace.
- Causality: $e_i \rightarrow e_j$ iff $e_i \succ e_j \wedge e_j \not\succ e_i$
- Unrelated: $e_i \# e_j$ iff $e_i \not\succ e_j \wedge e_j \not\succ e_i$ in the same trace, has to hold for all traces.
- Parallel: $e_i \parallel e_j$ iff $e_i \succ e_j \wedge e_j \succ e_i$

An indirect relation between two or more unrelated events may then arise. If $e_i \rightarrow e_j \wedge e_i \rightarrow e_k \wedge e_j \# e_k$, both e_j and e_k are preceded by e_i, but are not related. This means that we have to choose between either e_j or e_k after we have seen e_i. This can be seen in Figure 3 with *register request* and *examine thoroughly, examine casually*. The discovered relations are then used to discover a Petri Net by making use of their corresponding patterns which can be seen in Figure 3. In this figure,

- *register request* is directly followed ($\rightarrow$) by *examine thoroughly, examine casually* and *check ticket*.
- *examine thoroughly, examine casually* and *check ticket* are directly followed ($\rightarrow$) by *decide*.
- *decide* is directly followed ($\rightarrow$) by *pay compensation, reject request* and *reinitiate request*.
- *reinitiate request* is directly followed ($\rightarrow$) by *examine thoroughly, examine casually* and *check ticket*.
- We choose (#) between *examine thoroughly* and *examine casually*.
- We choose (#) between *pay compensation, reject request* and *reinitiate request*.
- We execute *examine thoroughly* or *examine casually* and *check ticket* in parallel ($\parallel$)

The α-algorithm does have some limitations however. For example, it has trouble with discovering loops of length one or two, where it just adds a separated transition in the model, which can fire at any possible time without restriction. Some other process discovery algorithms are: *Inductive Miner, Heuristic Miner* and *Split Miner*, which will be discussed in Section 3.2.

2.3.6 Which quality criteria should be taken into account? [19]

When a model is created, the following four criteria should be taken into consideration

- *Fitness* The discovered model should allow for the behaviour seen in the event log.
- *Precision* The discovered model should only allow for little additional behaviour compared to the log.
- *Generalization* The discovered model should only allow for additional behaviour from the data generating process, or system, that produced the event log.
- *Simplicity* The discovered model should be as simple as possible.

When generating a process model, this should have a well defined and unambiguous language. The model also should be sound [12], since there should not be unexecutable steps in a model, or e.g., customers waiting in a deadlock.

When generating such a process model, having high fitness is a desirable property. This means that the behaviour in the log can be executed by the model. Log-conformance measure *fitness* describes the relative share of fitting and unfitting behaviour. Fitness is 1 if all behaviour is fitting, i.e., all behaviour in the event log is represented in the model, and 0 if all behaviour is unfitting.

Another desirable property is high precision. A model has high precision when the behaviour of the model is present in the event log. Log-imprecise behaviour is behaviour that is included in the model, but not observed in reality. If the model and the log contain neither unfitting nor log imprecise behaviour, the model is equivalent to the log. Log-conformance measure *log precision* denotes the part of behaviour in the event log that is precise: if all behaviour of the model is present in the event log, then precision is 1. In process discovery, the assumption is made that the event log does not contain all behaviour, the event log only contains examples. If all possible behaviour is assumed to have been recorded, one could just use the event log as a prediction of future behaviour. Therefore, in many use-cases, prefect precision is not achievable. Notice that models and systems might have an unbounded number of traces, e.g., in case of loops, while event logs are always bounded. Thus, an event log can never contain all behaviour of a system that allows for indefinite execution. Therefore, a model with a loop can conceptually never be perfectly log precise. This challenges log measures, as these measures need to quantify how much of the unbounded behaviour in the model is used in the bounded event log as well.

Log-conformance measure *generalization* aims to estimate recall from the system. Recall is the amount of behaviour that is in the system, which is usually

unknown, which is also in the model. Generalization aims to indicate the likelyhood that future behaviour will be represented by the model, and computes this estimate using only the event log. Generalization techniques typically measure some property of the combination of the event log and the model, for instance the variety of the event log. If the event log contains the same behaviour multiple times, then generalization is assumed to be high. It is likely that a future trace is already present in the event log, and thus executable by the model. However, if every trace in the event log is different, then generalization is low, as future traces will probably not already be in the event log, and thus unexecutable by the model.

The log-conformance measure *simplicity* is intuitively defined as the understandability of a process model by a human analyst, which is a highly subjective and ambiguous definition. To measure simplicity, typically circumstantial, presumably complicating factors can be considered, such as the size of the model, partitionabilty, cyclicity, concurrency or, redundant model elements, and all these factors can be measured in several ways. Furthermore, simplicity could be measured with respect to an event log or system, i.e., it could express whether the model is a complex representation of an event log or system.

Usually there is a trade-off between these four criteria. When a model has high fitness, it can replay most of the traces in the event log. Precision is related to the notion of *underfitting*, in the sense that when a model has poor precision, it allows for behaviour that is very different from what was seen in the event log. Generalization on the other hand is related to the notion of *overfitting*. An overfitting model does not generalize enough and only allows for behaviour seen in the event log, it is driven by examples found in the log. The last criterium is related to Occam's Razor, which states that "one should not increase, beyond what is necessary, the number of entities required to explain anything". Following this principle, we look for the simplest process model that can explain what is observed in the event log. It turns out to be challenging to balance these four criteria. The obvious trade-off between underfitting and overfitting is immediately observed. Also, an oversimplified model is likely to have low fitness or lack of precision.

3 Related Work

Data mining is the process of discovering patterns in large data. Exceptional Model Mining is a data mining technique, which wants to discover relations between items, to create subsets which show interesting behaviour. Exceptional Model Mining is not the only technique that can be applied to discover relations in a dataset. Other techniques do so as well using different strategies based on different notions of interestingness. The techniques *Subgroup Discovery, Frequent Itemset Mining, Association Rule Mining, Emerging Pattern Mining, Contrast Set Mining* and *Skypattern Mining* want to achieve similar results. A more in-depth explanation of these techniques can be found in Section 3.1

Process Mining can be seen as the bridge between data mining and model-driven Business Process Management (BPM). Process Mining sits on the data mining abstraction level, in the way that *Process Mining is the process of discovering and visualising a process*, for which various techniques exist. As seen in Section 2.3, Process Mining operates on Event Data, to generate process models, using a process discovery algorithm. These process models are then repaired by comparing them against the event log using alignments. Each of these steps comes with their own problems, for which solutions have been developed. Some of these are *Model Repair, Alpha Algorithm, Inductive Miner, Split Miner* and *Concurrency Detection.* A more in-depth explanation of these techniques can be found in Section 3.2

Discovering relations between cases, and figuring out which cases behave differently compared to the majority of cases are Process Mining problems which show similarities to the problem Exceptional Model Mining tries to solve. Some of these techniques are *Clustering, Pattern Mining, Anomaly Detection* and *Subgroup Discovery for Process Mining* A more in-depth explanation of these techniques can be found in Section 3.3

3.1 Exceptional Model Mining

3.1.1 Subgroup Discovery

As already seen in Section 2.2.1, Subgroup Discovery is a data mining technique that discovers interesting associations among different variables with respect to a property of interest [14]. It tries to discover relations between properties, variables or attributes of a set with respect to a target variable. These subsets can be found by comparing the distribution of this attribute in such a subgroup and in the entire dataset and checking whether these distributions differ. These subsets are described in the form of individual rules, where a rule, which describes the discovered subgroup, can be defined as:

$$R : Cond \rightarrow Target_{value}$$

Where $Target_{value}$ is the value of the target attribute of interest for the discovery task. Subgroup discovery is related to Exceptional Model Mining, as it allows for the discovery of interesting subsets based on a user selected target attribute. A limitation of SD is that the assessment of a subgroup's exceptionality can be based on only one single attribute having a deviating distribution. Exceptional Model Mining was introduced to tackle this problem. Exceptional Model Mining allows for the discovery of subgroups based on a more complex target model, which contains multiple target attributes. This makes Exceptional Model Mining a generalization of Subgroup Discovery.

3.1.2 Frequent Itemset Mining and Association Rule Mining [22]

A frequent itemset is a set of items that appears at least in a prespecified number of transactions. Frequent itemsets are typically used to generate association rules. Frequent Itemset Mining operates using Support and Frequency. Suppose we have set of items I, from which we can select an itemset X, such that $X \subseteq I$. Also assume we have a transaction database T, in which transactions are stored using a transaction ID (t). The Support is the frequency of X in all transactions $tid \in D$.

$$Support(X, T) = \frac{|t \in T; X \subseteq t|}{|T|}$$

The Frequency of an itemset X in T is the probability of X occuring in a transaction $t \in T$:

$$Frequency(X, T) = P(X) = \frac{Support(X, T)}{|T|}$$

In other words, Frequent Itemset Mining tells us something about which items are frequently bought together and in which transactions this happened. For example, this allows stores to change their shelves, update their catalogs, or increase suggestions. It relates to Exceptional Model Mining in the way that it allows for the discovery of subsets of items frequently selected together, even though these relations might not have been obvious and thus are interesting.

Association Rule Mining is a popular data mining technique because of its wide application in retail and marketing communities as well as other more diverse fields. Association Rule Mining is a method of discovering relationships of the form $X \rightarrow Y$ amongst itemsets that occur together in a database, where X and Y are disjoint itemsets. Support and Confidence measures serve as the basis for customary techniques in Association Rule Mining. These measures are pre-specified by the users, such that rules that are not interesting or useful enough, are not considered. Association Rule Mining discovers transactions which include X then are also likely to include Y. If X has a support of 10% means that only 10% of the transactions contain X. The Support of an itemset X is the same as the Support defined above. The Confidence of an association

rule $X \rightarrow Y$ is the ratio between the transaction that contain $X \cup Y$ and the transactions that contain X

$$Conf(X \rightarrow Y) = Support(X \cup Y)/Support(X)$$

In other words, Association Rule Mining allows for the discovery of relations between items that are frequently bought together. It can discover how likely it is that someone selects item Y, after item X has already been selected. For example, this allows webshops to make suggestions based on items that previously have been purchased by the customer. It relates to Exceptional Model Mining in the way that it allows for the discovery of subsets of items frequently selected together, even though these relations might not have been obvious and thus are interesting.

3.1.3 Contrast Set Mining [16]

Contrast Set Mining is the task of finding contrast sets as "conjunctions of attributes and values that differ meaningfully in their distributions across groups". It is a form of Association Rule Mining, but differs in the sense that instead of finding rules that describe the current situation, it discovers rules that differ meaningfully in their distribution across groups. This means that they can be used as predictors for those groups.

Assume we again have a set of items I, and that we have a set of mutually exclusive user-defined groups G. For every contrast itemset $X \subseteq I$, we can calculate their support for every $g \in G$. We can now use these support values to predict which itemset X we will encounter, given group g. Contrast set discovery seeks to find all contrast sets whose support differs meaningfully across groups. Once all significant (Eq. 1) and large (Eq. 2) contrast sets are found, an 'interesting' subset should be presented to the end user. Formally,

$$P(X|g_i) \neq P(X|g_j) \tag{1}$$

$$SuppDiff(X, g_i, g_j) = |Support(X, g_i) - Support(X, g_j)| > \delta \tag{2}$$

where δ is the *minimum support-difference*. The significance is the basis of a statistical test of 'meaningful', and being large is a quantitative test thereof.

In other words, Contrast Set Mining tries to discover the differences between groups, where groups are determined by a attribute that distinguishes them. It allows for the prediction of attributes items will have, based on the group they belong to. For example, this allows shops to give suggestions on the product the customers would like, given what the goal of their visit is. It relates to Exceptional Model Mining that it allows for the creation of subgroups based on a target attribute (group classifier). The results are groups that statistically differ when compared to the rest of the dataset.

3.1.4 Emerging Pattern Mining [15]

Emerging Pattern Mining is similar to Contrast Set Mining, in the sense that it uses the support for an itemset X in group g_i as its classifier, but uses it in a different way. Contrast Set Mining uses the *difference* of support between groups to determine which itemset X is most likely to appear, given a group g_i. Emerging Pattern Mining uses the ratio of support in two datasets, to capture emerging trends in time-stamped data, or useful contrasts between data classes. From a semantic point of view, emerging patterns are association rules with an itemset in the rule antecedent, and a fixed consequent: $X \rightarrow D_1$', for dataset D_1 being compared to another dataset D_2. The measure of quality of emerging patterns is the *growth rate*, or ratio between the two supports. A pattern with 10% support in D_1 and 1% support in D_2 is better than a pattern with 70% support in D_1 and 10% support in D_2, since $\frac{10}{1} > 70/10$. From the association rule perspective, $GrowthRate(X, D_1, D_2) = \frac{Confidence(X \rightarrow D_1}{1 - Confidence(X \rightarrow D_1)}$. Growth rate provides an identical ordering to Confidence, except that growth rate is undefined when Confidence = 1.0.

In other words, Emerging Pattern Mining looks for the increase in support between two datasets using the Support ratio. This relates to Exceptional Model Mining because it allows for the discovery of itemsets, based on a target attribute (group classifier). The results are groups that statistically differ when compared to the rest of the dataset.

3.1.5 Skypattern Mining [26]

Skypattern Mining tackles the problem of extremely large output for pattern mining operations. Which in worst case can be exponential to the number of items in the dataset. One solution makes use of a threshold, which requires domain specific knowledge. Another solution is to include user preferences, e.g., only keep the *top-k* scoring results. An issue with this approach, is that it heavily depends on the dataset size and the selection of k. Skypattern Mining tries to make results *useful from a user-preference point of view.*

Skypattern Mining bases its decision making based on *"Pareto efficiency or optimality queries"*. This means that point a dominates point **b** if a is better than b in at least on dimension, and not worse in any other dimension. For example, a user selecting a set of patterns may prefer a pattern with a high frequency, a large length and a high confidence. In this case, we say that pattern a dominates another pattern b if $frequency(a) \geq frequency(b), length(a) \geq length(b), confidence(a) \geq confidence(b)$, where at least one strict equality holds. Skypattern Mining thus returns the patterns that are not dominated by any other.

Skypattern Mining relates to Exceptional Model Mining because it allows for the discovery of subsets that contain deviating behaviour based on user-specified

constraints. It differs in the fact that these constraints are target attributes, refinement depth and quality measures for Exceptional Model Mining, where for Skypattern Mining, support, confidence and size are possible constraints.

3.2 Process Mining

3.2.1 Model Repair [10]

If the model does not describe the behaviour in the log (skips an activity), one could choose to create a new model that does, but then this model will not be similar to the original process model. That is why *Model Repair* has been introduced.

Model Repair allows the user to *repair* the original model, to replay most of the event log, while staying close to the reference model. Model Repair alters small part of the original model, such that the event log has a better replayability. The three main reasons for model repair are *improving conformance checking diagnostics, monitoring process evolution* and *supporting customization.* It is important that the *repaired model remains as close to the original model as possible.*

Model Repair focuses on *repairing control-flow to fit an event log.* It does so by computing an alignment for each case between the event log and the process model and determining where the model and the trace deviate. These mismatches are then used to update small parts of the model, such that it can now replay the event log.

Model Repair's main focus is on improving an already existing process model, to conform more to the event log, rather than generating a new one. For this to happen, we need an already existing process model and we need to analyse the behaviour found in the event log. Both can be seen as prior process knowledge, which we already know to be hard to obtain. It also only improves an already existing view on the execution of the process, rather than discovering bottlenecks that slow the execution of that process.

Model repair could be helpful, if in the discovered subset of traces using Exceptional Model Mining, there still would be outliers. The effect of these outliers on the behaviour of the generated model could be adjusted using Model Repair.

3.2.2 Concurrency Detection [3]

Concurrency Detection allows for the discovery of events in traces that are executed in parallel. If an event log contains the traces $\langle ..., A, B, ... \rangle$ and $\langle ..., B, A, ... \rangle$, events A and B are classified as concurrent. This classification is called *global concurrency.* This classification states that A must either be preceded or followed by B, regardless of where the events occur in the log. In practice, this

property does not always hold. Consider the traces $\langle C, D, E, F, G, H, L, A, B\rangle$ and $\langle B, A, C, D, E, F, G, L\rangle$. These traces have the same order for *C,D,E,F,G,H,L* but the order between A and B depends on where these activities occur. *Global concurrency* detection will classify A and B as concurrent, while this is not the case. Global concurrency tends to over-generalize the behaviour captured in the log, by allowing the execution of sets of event types in any order throughout the log, even when such activities are in fact not concurrent, or only in certain parts of the log. *Local concurrency* detection allows for the discovery of the relation between event types in certain execution states of the process. It does so by constructing a state transition graph from the event log and traverse this graph to detect concurrency relations in between pairs of states.

Local concurrency detection is a nice feature that could be used to analyse the subsets of traces we obtain using Exceptional Model Mining. We can then detect concurrency on a local level for the subset and the complement of the subset, to check whether there are patterns that differ between both. This would make further analysis more accurate. It allows the user to learn more about the differences, without the need for base process knowledge. The obtained results can, together with the discovered subsets, be discussed with the process owner.

3.2.3 Heuristics Miner [28]

The Heuristics Miner is a framework that allows for the generation of process models. For its generation, it only considers the order of events within a trace, the order of events among cases is not important. For every trace, the order of events is determined using the timestamp. Using the case identifiers and timestamps, an event log is obtained. An event log is a multiset of traces, thus traces can occur more than once. For every pair of events in a trace, their relation is discovered. The following relations can be discovered using the Heuristics Miner:

- $a > b$ if there is a trace $\langle .., a, b, ...\rangle$
- $a \rightarrow b$ if $a > b$ and $b \not> a$
- $a||b$ if $a > b$ and $b > a$
- $a\#b$ if $a \not> b$ and $b \not> a$
- $a >> b$ if $\langle ..., a, b, a...\rangle$
- $a >>> b$ if $\langle ..., a, ...b, ...\rangle$

The first four relations are the same as used with the $\alpha - algorithm$. The $\alpha - algorithm$ does not take the frequency of traces into account. This can result in erroneous results when the log is noisy. If we have 1000 traces where $a > b$ and one traces where $b > a$, the alpha algorithm cannot find a correct conclusion.

To tackle this problem, the Heuristic Miner constructs a dependency graph, where a frequency metric is used to determine how certain we are that there truly is a dependency between a and b ($A \Rightarrow B$. This dependency is calculated as follows,

$$A \Rightarrow B = (\frac{|a > b| - |b > a|}{|a > b| + |b > a| + 1})$$

This value is always between -1 and 1. For every pair of events, the dependency value is calculated and a matrix storing these values is generated. We know that every non-initial information must have at least one causal activity, and every non-final must have at least one dependent activity. Using this information in the so called *all-activities-connected heuristic*, we can take the best candidate (with the highest $a \Rightarrow b$ value). For every event in the matrix, we can now select the best candidate and generate the dependency graph. For the lower frequency dependencies, it is uncertain whether these lower frequencies are caused by noise, because it is a low frequency pattern. The *Dependency threshold, Positive observations threshold* and *Relative to best threshold* have been introduced to tackle this problem. These allow for the selection of relations between events if the relation values are better than the specified threshold.

The Heuristic Miner can also discover short loops (1,2), AND/XOR split/joins and non-observable tasks(3).

$$a \Rightarrow a = (\frac{|a > a|}{|a > a| + 1}) \tag{3}$$

$$a \Rightarrow_2 b = (\frac{|a >> b| + |b >> a|}{|a >> b| + |b >> a| + 1}) \tag{4}$$

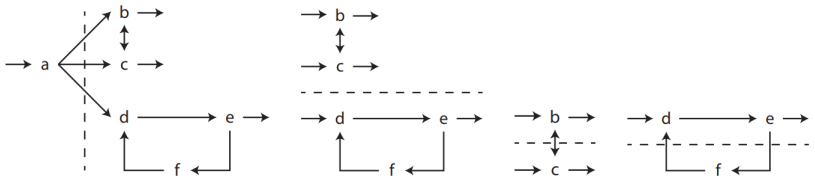
$$a \Rightarrow b \wedge c = (\frac{|b > c| + |c > b|}{|a > b| + |a > c| + 1}) \tag{5}$$

The dependency graph calculated earlier shows the causal relation between events. For the events that follow one event, their corresponding relation needs to be calculated. If the causal relation $a \Rightarrow b \wedge c = 1$, b and c have a AND-relation. If the relation has a value below the $\wedge - threshold$, b and c have a XOR-relation. These discovered relations are then used to generate the process model, corresponding to the log.

The Heuristics Miner can be used to generate process models from the discovered subsets of traces using Exceptional Model Mining, to further analyze the results and be able to discuss them in a clear and understandable way with the process owners.

3.2.4 Inductive Miner [18]

The Inductive Miner is a framework that makes use of block structured process models that guarantees to return sound and fitting process models. It uses any



(a) $G(L)$. (b) $G(L_2)$ (c) $G(L_3)$. (d) $G(L_4)$.

Figure 4: Several directly-follows graphs. Dashed lines denote cuts.

flavour of block-structured process models: new blocks/operators can be added without changing the framework and with few proof obligations. The framework uses a divide and conquer approach to decompose the problem of discovering a process model for a log L into discovering n subprocesses of n sublogs obtained by splitting L.

Given a set of process tree operators $\bigoplus$, a framework B is defined to discover a set of process models using a divide and conquer approach. Given log L, B searches for possible splits of L into smaller $L_1, .., L_n$, such that these logs combined with an operator $\oplus$ can produce L again. It then recurses on the found decisions and returns a cartesian product of the found models. The recursion ends when L cannot be divided any further.

The directly-follows relation is used to generate the *directly-follows graph* of L. It is a directed graph containing as nodes the activities of L. An edge (a,b) is present if and only if trace $\langle .., a, b, ... \rangle$ exists in L. The directly-follows graph is split into subgraphs by discovering the 'dominant' operator. For example, the first graph in Figure 4a can be partitioned into two sets of activities as indicated by the dashed line such that edges cross the line only from left to right. This pattern corresponds to a sequence where the activities on the left side of the line precede the activities on the right side of the line. This is the decisive hint on how log L should be split. The entire graph can be split into n sets of nodes with characteristic edges in between. Log L can then be split according to the identified operator, and the framework recurses on each of the split logs.

The Inductive Miner can be used to generate process models from the discovered subsets of traces using Exceptional Model Mining, to further analyze the results and be able to discuss them in a clear and understandable way with the process owners. Since it allows for the discovery of models with frequency and performance information, we used this miner to generate the models from the subsets we discovered. These results are discussed in Section 6.4.3.

3.2.5 Split Miner [4]

The Split Miner is able to generate deadlock-free and sound models from an event log. It starts by generating the directly follows graph, which is then analysed for short-loops and self-loops. Concurrency is shown as arcs from a to b and$b$ to a, meaning that causality and concurrency are mixed up. Whenever a likely concurrency relation is discovered, this edge is removed, which results in a *pruned directly-follows graph* (PDFG). This concurrency is discovered by analyzing the ratio between $a > b$ and $b > a$. If this ratio is below a specified threshold, we assume the events to be concurrent.

Next a filtering algorithm is applied on the PDFG to strike balanced fitness and precision, with low control-flow complexity. This filtering is done based on three properties. Namely, every node must be on a path from the source to the sink. This property ensures a deadlock-free model. For each node, its path from source to sink must be the one having maximum capacity. In this context, maximum capacity is the frequency of the least frequent edge of the path. This property maximizes fitness. Finally, the number of edged on the PDFG must be minimal. This property maximizes fitness.

In the fourth step, split-gateways are discovered for events with more than one outgoing event in the filtered PDFG. These splits are generated while relying on the concurrency relations discovered in the second step of the algorithm. Tasks that directly follow a split gateway are concurrent to the set of tasks that does not directly follow the same split. When it is known which tasks are successors of the same gateway, the gateway type can be determined by checking whether its successors are concurrent or mutually exclusive.

Once the split gateways have been placed, the join gateways can be discovered. When discovering these gateways, the number of OR-joins has to be minimized, to increase the understandability of the process model.

Split Miner is another process discovery algorithm that could be used to generate process models from the discovered subsets of cases using Exceptional Model Mining. These models can again be used to discuss and verify the obtained deviations with the process owners.

3.3 Pattern finding in Process Mining

3.3.1 Clustering for Process Mining [23]

Due to the diverse and unpredictable nature of most processes, it is very hard to discover the actual process, since so many deviations exist. The discovered models will usually be very unstructured and unreadable, also called “spaghetti models”. To tackle this problem of diversity, one can reduce the number of cases which are analysed at once. This can be done by measuring similarity of cases and use this information to divide the set of cases into more homogeneous

subsets.

Every clustering algorithm tries to group sets of similar points. For trace clustering, these points are log traces. Traces that show similar behaviour should be grouped together. To determine this similarity, *profiles* are used. These profiles are sets of related *items*, which describe the trace from a certain perspective. Every item is a metric, which assigns to each trace a specific numeric value. A profile with n items, can be seen as a vector $\langle i_1, .., i_n \rangle$. These vectors can subsequently be used to calculate the distance between any two traces, using a distance metric. An example of such a profile is the *activity* profile, which, for every trace, counts the occurrences of all activities in that trace. Other examples are the *resource* profile, which counts the resources that executed events in the trace, or the *transition* profile, which contains all *directly follows relations* in the trace. Using a distance measure on these profiles, we obtain the relative distance between traces. The obtained distance can then be used with a clustering algorithm (e.g.*k-means*) to obtain clusters of traces that show similar behaviour.

The ability to group traces that show similar behaviour is a desirable property to reduce the information overhead that comes with the usually diverse nature of processes. The resulting trace clusters can be further analysed, to discover their behaviour. If one cluster contains most of the traces, we can with a certain degree of certainty, assume that this is the desired process execution. The traces in other clusters contain different behaviour, which might be worth investigating. However, this still requires a fair amount of user interaction and process knowledge. The difference between trace clustering and Exceptional Model Mining is that Exceptional Model Mining does not need this base knowledge to discover deviating subsets. The only base knowledge it requires are interesting attributes, which are used to discover the most interesting subgroups. Exceptional Model Mining could also be applied after trace clustering, to compare the clusters against each other and verify whether on or more of these clusters contain deviating behaviour, compared to the others. This requires a lowering of the minimum support value, since there are fewer items to evaluate.

3.3.2 Pattern Mining for Process Mining [24]

The goal of Pattern Mining is to detect rules that describe patterns in the dataset at hand. An example of this is a market basket analysis, which discovers items that are frequently purchased together. An interesting perspective in pattern mining is detecting unexpected patterns. These patterns might be an interesting starting point for further research. Sequential pattern mining is one of the variations on pattern mining, which goal is the discovery of sequential patterns, or, events that happen in sequence.

This seems like an interesting ability to have within Process Mining, as it allows for the discovery of patterns in traces which occur frequently. However, the

issue with sequential pattern mining already shows through its name, namely that it discovers *sequential* patterns. This means that it does not have a notion for loops or concurrency, features that are occur frequently in processes. That is why Local Process Model mining has been brought to life. Local Process Model mining (LPM) allows for the discovery of patterns that occur frequently in the traces in the event data, without looking at these traces as a whole. It is also capable of detecting concurrency and loops, which makes it more valuable than sequential pattern mining. This allows for the creation of correct process models that show only part of the process, and not the entire process.

LPM is a technique that allows for the detection of events that frequently occur together, be it sequential, concurrent, or in a loop, over a dataset. This is an interesting ability to have, as it allows for the detection of process steps that happen frequently together. It then becomes interesting to check whether these detected patterns show behaviour that is unexpected, which means that something interesting is happening. It does however not allow for a user induced notion of interestingness, so the user still has to investigate the discovered pattern to check whether he deems them interesting.

The difference between LPM and Exceptional Model Mining is that LPM requires base knowledge of the process to be able to distinguish between the cases that are executed correctly and the cases that contain the deviating behaviour. Again, this knowledge is not needed when applying Exceptional Model Mining. It would be useful to use LPM on the discovered subsets of traces and their complements, to check whether there are patterns in the traces that differ between the two subsets. The knowledge of the existence of these patterns can be used as an extra verification that the discovered results were correct, and to discuss with the process owners, to inform them that the process is executed differently for the subsets.

3.3.3 Anomaly Detection in Process Mining [13]

Anomaly detection is the identification of rare observations or events, which do not match with the majority of the data. This is strange and probably undesired behaviour, since the majority of cases in a process usually show the desired or normal functionality. Detecting such anomalies is considered valuable since they show exceptional behaviour, which might be interesting to explore.

Anomaly detection is also possible in Process Mining, and it allows for the detection of cases that show behaviour which does not match with the desired business process. These deviations then can be validated with the process owner to check whether they are unexpected or expected exceptions (e.g. an emergency in a hospital). Detecting these anomalies can thus be very helpful, however, due to the diverse nature of business processes, there will be many deviations which only occur once or twice. They still are anomalies, but due to their infrequent nature, they are not very interesting to explore. The interesting anomalies are

the ones that occur regularly, so these frequent exceptions are what we want to discover. To solve this problem the original authors proposed a solution which works based on subgraphs. The authors use the event data and a process model to detect frequent subgraphs and then apply conformance checking to validate which of these subgraphs show exceptional behaviour.

Anomaly detection is thus able to detect anomalies within a process, which might require further investigation. The drawback of this method is that an already known to be correct process model has to be used to detect the anomalies. To have such a process model, it either has to be obtained through the process owners or be discovered, which requires a good understanding of the event data at hand. The difference between Exceptional Model Mining and anomaly detection is thus that Exceptional Model Mining is able to discover these deviating cases without having any prior process knowledge.

3.3.4 Subgroup Discovery for Process Mining [11]

Subgroup discovery can be used to discover groups that show similar behaviour when compared to a target attribute. This data mining task has recently gained an application for Process Mining. The authors of the research show the possibilities of applying subgroup discovery in Process Mining, which is most similar to what we are currently researching. Since this paper is so relevant, the next Section will be a more in-depth explanation on the research Wil van der Aalst et al. performed. The research performed in this paper investigates the possibilities of generating subgroups which contain cases that contain interesting behaviour based on the user specified target attribute. In many applications, stakeholders prefer to analyse and know more about a subset of cases rather than all cases. Some examples are:

- Deviating cases from the reference model
- Cases with high or low performance
- Cases with high profits for the company
- Unfinished or canceled cases
- Cases from a particular period
- Cases that pertain user complaints
- Events related to particular products or services

When investigating such subsets, it is important to see which attributes they have in common. If the user would, for example, discover that cases that take a deviating amount of time to complete are all handled by one department, this could be an indication that something is wrong within this department. Little to no research has been performed to extract this kind of information from the event logs. The main goal of the performed research is to apply subgroup

discovery techniques in the Process Mining domain, to discover the statistical most interesting patterns in a subset of cases called the target groups. Subgroup discovery deviates from the previously described clustering in the sense that subgroup discovery assumes that there is already a group of samples that already has a class label (e.g. deviating or not). In other words, subgroup discovery searches for groups in the dataset that have characteristics in common that are not appearing as often in the other cases.

Most Process Mining techniques consider events as the starting point for analysis, whereas with this architecture the focus lies on cases and their corresponding properties. To be able to apply subgroup discovery in Process Mining, Adaptions to the available data had to be made. Starting with an event log, subgroup discovery alters this event log to show the properties of all individual cases. This allows for the possibility to describe each case with their characteristics. Next, all cases and their characteristics are combined to obtain a *case base*. After this a class attribute and its desired properties or values are selected. This class attribute can be seen as the target attribute in subgroup discovery. Afterwards the case base is split. In the *target group* the cases are placed that contain the desired properties for the class attribute. All other cases are placed in a different subset. Finally, they apply subgroup discovery on the target group to obtain many different subgroups for which the class attribute has the same properties. These many different subgroups are all different based on their size, interestingness, distribution and their effects on the target group. They used an *unusualness* measure to compute the interestingness of each subgroup on the target group. For these subgroups it is possible that they have more than one descriptive attribute, which is a desired property of general subgroup discovery. After discovering these interesting subgroups, it is possible to use these subgroups for further analysis to obtain more insightful results in the business process.

The authors implemented their ideas and created a framework in ProM [27]. This tool was then used on a real subset for which they discovered subgroups which showed interesting and divergent behaviour that they were unable to discover using different methods. This tool (1) shows the user how the subgroups impact the target group, (2) the change in percentage of classes in a subgroup compared to the whole class base, (3) the number of samples in each subgroup and (4) a table with measured values for coverage, support, and confidence for each subgroup.

This topic already is a giant leap in the direction we are interested in. It allows to create subgroups from event logs based on a target attribute described by descriptive attributes. However, it still does not allow the creation of subgroups based on a more complex target model based on multiple target attributes, so there is still room for improvement. In the research paper it is mentioned that Exceptional Model Mining has this possibility, but they did not explore this any further in this or other literature.

4 Methodology — Exceptional Model Mining for Process Mining

Now that both topics of EMM and PM have been introduced, the goal of using EMM for PM might still be ambiguous. This Section will try to clarify the end goal.

As seen in Section 2.3, the goal of Process Mining is to create an easy to understand Process Model to find bottlenecks or deviations in the desired process to increase the workflow and get as close to the happy flow as possible. The discovery of these bottlenecks can be a long and tedious task, which requires a decent base of knowledge of the process under investigation. This knowledge can be obtained through interaction and communication with the process owners, but this can be a difficult task, since usually process owners do not know where the problems in their process reside. They can provide information on how the process is supposed to be executed, which can then be used to analyse the event log to discover traces that contain deviating behaviour. Another option is to obtain the model from the event log, by only including the most occurring trace-variants when creating the model. The Process Model then shows the main behaviour described in the log. Both methods are convenient for understanding the process, but will most of the time not show the deviations which cause bottlenecks, which will provide valuable information.

The goal of Exceptional Model Mining is to find subgroups which show divergent behaviour based on multiple attributes. Exceptional Model Mining splits the dataset at hand in two subsets, the descriptive and target attributes, where the descriptive attributes are used to describe the discovered subgroups, and the target attributes are used to evaluate the discovered subgroups using a user-specified quality measure.

When we can use Exceptional Model Mining to obtain divergent subgroups from (adapted) Event Data, we are able to gain insights on divergent behaviour residing in the Event Data which are described by the descriptive attributes.

4.1 What do we want to achieve doing this?

When we use Exceptional Model Mining to discover subgroups which show divergent behaviour based on our target attributes when compared to the rest of the log, we can analyse the dataset of those subgroups and try to figure out what is causing this deviating behaviour. This way we can obtain interesting behaviour which resides in the log, without really having to understand the underlying process.

We need to prune the dataset and look at all attributes, since the description might not be very helpful. This problem resides in the fact that the user

chooses a *search depth* d which declares an upper bound on the length of the descriptions. This description then does indicate an interesting subgroup, but the descriptive attributes might not be very helpful. If we then prune the dataset using this description, we will get a dataset which adheres to this description, but also shows the other attributes, which might give a better idea of a specific set of attributes that causes the deviating behaviour as well, since it describes the same subgroup.

When we have obtained an idea on the whereabouts of a possible cause of the deviations, we can then use Process Mining techniques to filter the Event Data based on these descriptions and then obtain a Process Model, which shows the process that corresponds to the behaviour shown in the event log for the obtained subgroup. We can also check whether the trace attributes corresponding to this log show odd behaviour, e.g. that the traces in this log have a very long running time. These obtained results can then be shown to and discussed with the process owner, to see whether these results are new and unexplainable, such that the process owner can further look into the issue.

4.2 How do we want to achieve this?

We want to change the event log in such a way that it is compatible with an Exceptional Model Mining Framework implementation. The event log stored in *XES* format is a nested list, where the first entry is the first trace in the log. An example of a trace is:

{'attributes': {'concept:name': '1', 'creator': 'Fluxicon Nitro'}, 'events':
[{'concept:name': 'register request', 'org:resource': 'Pete', 'time:timestamp':
datetime.datetime(2010,12,30,11,2), 'Activity': 'register request', 'Resource':
'Pete', 'Costs': '50', 'case:concept:name': '1', 'case:creator': 'Fluxicon Nitro'},
{'concept:name': 'examine thoroughly', 'org:resource': 'Sue', 'time:timestamp':
datetime.datetime(2010,12,31,10,6), 'Activity': 'examine thoroughly',
'Resource': 'Sue', 'Costs': '400', 'case:concept:name': '1', 'case:creator':
'Fluxicon Nitro'},
{'concept:name': 'check ticket', 'org:resource': 'Mike', 'time:timestamp':
datetime.datetime(2011,1,5,15,12), 'Activity': 'check ticket', 'Resource':
'Mike', 'Costs': '100', 'case:concept:name': '1', 'case:creator': 'Fluxicon
Nitro'},
{'concept:name': 'decide', 'org:resource': 'Sara', 'time:timestamp':
datetime.datetime(2011,1,6,11,18), 'Activity': 'decide', 'Resource': 'Sara',
'Costs': '200', 'case:concept:name': '1', 'case:creator': 'Fluxicon Nitro'},
{'concept:name': 'reject request', 'org:resource': 'Pete', 'time:timestamp':
datetime.datetime(2011,1,7,14,24), 'Activity': 'reject request', 'Resource':
'Pete', 'Costs': '200', 'case:concept:name': '1', 'case:creator': 'Fluxicon Nitro'}]

This trace has as as trace attributes the name of the trace (*concept:name*) and the trace creator (*creator*). It also contains five *events* (*concept:name, Acitivity, register request, examine thoroughly, check ticket, decide, reject request*,

	case:concept:name	concept:name	org:resource	time:timestamp	Activity	Resource	Costs	case:creator
0	1	register request	Pete	2010-12-30 10:02:00+00:00	register request	Pete	50	Fluxicon Nitro
1	1	examine thoroughly	Sue	2010-12-31 09:06:00+00:00	examine thoroughly	Sue	400	Fluxicon Nitro
2	1	check ticket	Mike	2011-01-05 14:12:00+00:00	check ticket	Mike	100	Fluxicon Nitro
3	1	decide	Sara	2011-01-06 10:18:00+00:00	decide	Sara	200	Fluxicon Nitro
4	1	reject request	Pete	2011-01-07 13:24:00+00:00	reject request	Pete	200	Fluxicon Nitro

Figure 5: A Trace transformed into a Matrix, for better readability

which each contain information on the person that executed it (*org:resource, Resource*), the execution time (*time:timestamp*), the cost of the execution (*Costs*) and the case creator (*case:creator*).

A trace has as first entry the trace attributes and as second entry the events with their corresponding attributes. When we want to view the first event of the first trace in an isolated way, we open the first entry of the first trace using indexing. Assume *log* is the name of our event log, the following operations are possible:

- *log*: Shows us the complete log, with each entry having an *attributes* entry, which contains the trace attributes and an *activities* entry, which contains the events and their attributes.
- *log[0]*: Shows us the first trace, which has an *attributes* entry, and an *activities* entry, each with corresponding values.
- *log[0][0]*: Shows the first event of the first trace in the log, with corresponding attribute-value pairs.

This format can be quite difficult to understand, since if an event contains many attributes, the list becomes pretty long, which makes it easy to lose track. It is possible to transform it into a matrix format, as seen in Figure 5. When observing this format, we see that every row in the matrix is an event, and every column of that row describes the event attributes. The names of these attributes are listed at the top, and the cell itself contains the value of the attribute name.

Since we are interested in traces and their events and attributes as a whole, spreading them over multiple rows is something we are not interested in. That is because this data format is not supported by the current Exceptional Model Mining framework, since this framework operates on datasets which are formatted like a *bag of records*, which is essentially a matrix, where each row is a separate entry, or item, with the corresponding columns being its attributes, or characteristics.

Thus the need arises for a conversion tool, which transforms an event log to a Case-Attribute matrix. In this matrix, each row contains a case, with each column containing the information needed to describe the trace. The trick here

is to make sure that as little information as possible is lost when transforming a trace.

5 Data preprocessing: from event log to case attributes data

Process Mining operates on event logs. These event logs contain traces, which contain trace attributes, the sequence of events, and the corresponding event attributes. When stored in a matrix format, these are stored as an event per row, with a case identifier and their other events in the corresponding columns. Exceptional Model Mining operates on datasets. These datasets are also stored in matrix format, where each row is an item which is described by its attributes stored in the corresponding columns.

5.1 How to convert the event log to be useful for Exceptional Model Mining?

We are interested in traces as a whole and observe that there is a mismatch between the two data formats. Exceptional Model Mining operates on single rows, which contain all information of one single instance in its column values. An event log spreads the information of a trace over multiple rows, where in each row an event and the corresponding attributes are stored. This way the sequence of events is clearly visible, and the information can be stored as clear and understandable as possible. Because of this mismatch, we cannot just take an event log and analyse it using an Exceptional Model Mining framework. It is possible, but then we would be able to create subgroups which contain divergent events. One of the situations where it would make sense, is if the user is interested in event completion times, but then the event duration would have to be explicitly stored as an event attribute. Subgroup Discovery, or Exceptional Model Mining using one attribute, could then be used to discover subgroups which have long completion times compared to the rest of the events. The discovered descriptions could then describe subgroups of events which have long completion times.

Single events are usually not the problem when a process is not as efficient as possible, so we need to look at complete traces. Therefore, we need to transform the event log in such a way that all trace attributes, events, and event attributes can be stored in a single row in the matrix. This way we can analyse the traces using the Exceptional Model Mining framework. For this transformation to be useful and valid, we need to take some things into consideration.

5.2 Which characteristics should be considered attributes?

Now that it has been discussed how the traces should be transformed, the actual transformation is still open for discussion. First we will describe the encodings that already have been developed in Section 5.2.1. Our design decisions, and the reasoning behind it, for which information we want to keep after the conversion process are given in Section 5.2.2.

5.2.1 Which encodings do already exist?

Various sequence encoding methods for storing traces in a single row have been developed. All of these are developed for when predictive analysis is the main goal. Predictive analysis refers to the act of making predictions about the future state of ongoing cases in a process [25]. The following sequence encoding methods are the most common:

- *Static.* This is how the case attributes, or *static* attributes are stored. They are the same for the entire trace and all events in that trace. This means that they can be added to the feature vector without any loss of information.

- *Last State.* This encoding method only uses the last available snapshot of the data. The size of the feature vector is proportional to the number of event attributes and is fixed throughout the execution of a case. The drawback is that it disregards all the information that happened in the past, using only the very last data snapshot. A solution is to include the last m states, thus increasing the feature vector m times. Since the feature vector size is fixed, the length of the trace does not matter.

- *Aggregation.* Last state encoding clearly suffers from information loss, where it neglects the data that has been collected during an earlier execution state of the trace. Another solution is to consider all events since the beginning of the trace, but ignore the order of the events. This encoding allows for the storage of information that an event attribute has been observed when executing the case. In particular, the frequency of the attributes is stored. Numerical values can be stored using general statistics, such as the mean, maximum or minimum and sum.

- *Index.* The index encoding uses all possible information (including the order) in the trace, generating one feature per each event attribute per each executed event (each *index*). This way, the original trace can be reconstructed based on its feature vector. The drawback of this encoding is that the length of the vector increases with each executed event, which means that this can only be applied to traces which all have the same length.

Since these encodings are all used in predictive analysis, they all operate under the assumption that the execution of a trace is not yet finished. This makes only keeping the last part of the trace sufficient, since the information in this snapshot can give enough information to make decent predictions. However, we are performing a descriptive analysis, which means that we want to keep as complete cases and as much information as possible. This is why we decided to design a conversion, or encoding tool, which is most similar to the *aggregation* encoding.

5.2.2 How did we design our conversion tool?

When we look at the explanation of a trace as described in Section 4.2, we quickly observe three trace characteristics that at least should be considered attributes.

- We need to know which *trace* is stored in which row. The most convenient way of doing this is using the row index as case identifier, where the first case identifier is used as the first index entry.
- The *case attributes* should be stored, since these are already stored at case level, we can just store them in separate columns for the correct trace. Since we iterate over the event log, case attribute frequencies will be equal to the number of events in the trace.
- The *activities* occurring in the trace should be stored, the most easy and obvious design decision is to iterate over the events in a single trace and count the occurrences of each activity per trace and use this as cell values. This way all activities and their occurrences are listed in one single row per trace.
- The *other event attributes* should be stored, this can also be done by iterating over the events and counting how often each attribute is encountered.

Now the most prominent and obvious attributes of each trace can be stored in one single row, making it thus compatible with the Exceptional Model Mining framework. The information we store and how we store it is similar to the information stored in [11]. The authors were able to obtain valid results using this information, so we assume it to be enough information.

These attributes can quite easily be stored without much loss of information. All information is stored, however, which attributes belong to which event cannot be retraced. This will most probably almost never be an issue, since the number of occurrences of an event attribute in a trace which is discovered in a subgroup also is an indication of the importance of that attribute. The user can then further investigate to which event these attributes belong by looking at the original event log.
One important piece of information that is lost this way is the sequence in which the events occur, as this is one of the most important pieces of information when evaluating traces in Process Mining. The order of the events and the relationship between events plays a significant role in Process Mining. Some ideas on how to resolve this issue, will be discussed in Section 7.3.

Another attribute that is useful to have knowledge about is the existence of loops in the trace, since loops indicate that certain events have to be executed more than once. If this is not the expected or desired behaviour, this is an indication that the process can be improved. This is a gain in execution speed

and thus a desired ability to have. At first, explicit mentions came to mind, but a loop usually is found when an event occurs more than once. Thus, when an event occurs more than once, as seen by the cell value of that event being larger than one in a trace, the user is pretty sure that a loop is present. It should however be noted that a loop is not always a bad thing, since it can also be a designed feature, where, e.g., a response to a request is always stored as the same type of event, and a response almost always has to happen more than once in a successful execution of the process. The importance of knowledge on the existence of loops is thus up to the end user to decide.

We know what attributes are desired to have when transforming an event log, but how this transformation should be done exactly, and how much user interaction is still required to do this, will be discussed in Section 5.3.

5.3 How much user interaction is required to transform the data?

It is now known which trace and event attributes are kept and how they should be stored. Now the actual transformation needs to happen. A Python implementation has been developed which is able to transform the event log to a Case Attributes table. It starts with an event log, stored in either trace format as seen in Section 4.2 or as seen in Figure 5. If the event log is stored as the first option, it transforms it to the table format using a transformation implementation from *Process Mining for Python (PM4PY)* [5]. This library assures that all information is kept when transforming the data. This transformation makes it easier to traverse the log, since the dataframe format in which it is stored allows for more complex operations.

The user then has to perform a dataframe fix, which fixes:

- *The time column*, by transforming it from a string value to a datetime value, which can be understood by Python.
- *The case order*, by storing them based on their case identifier in ascending order. This alteration is required, because our data transformation implementation iterates over the event log row by row and counts all event attributes.

This fix is done by calling the *dataframe_fix* function, which uses the log as dataframe, the case ID and time columns, and the desired timezone as input.

Next the user has to transform the event log such that it has the Case Attribute format. This can easily be done by calling the *case_attribute* function which takes the log as dataframe and an optional ignore list, which can be used to drop redundant or duplicate columns from the log (e.g., sometimes the resource is listed as both org:resource and as Resource, where only one is required). This choice is for the user to make, and does not change functionality. This function

then creates a row per case, in which the counts of the trace attributes, events, and event attributes are stored. It also adds the following columns:

- *start_date*, where the date of the first event in the trace is stored.
- *end_date*, where the date of the last event is stored.
- *duration*, where the duration is listed in (days, remaining time) format.
- *durHours*, where the duration is stored in hours.
- *Average_Event_Duration*, where the average duration per event is stored.
- *AEDIH*, where the Average Event Duration is stored In Hours.

These added features were very useful for our analysis goal, which focuses on deviations in the performance between subsets of cases. However, since improving performance is always a desired ability, these features will always be generated. The user can later decide to not consider these during the subgroup discovery process. When this operation completes, the user is left with the Case Attributes table, which is stored in dataframe format, of which an example can be seen in Figure 6. In this figure, we can see the number of times each event attribute is found per trace, and the columns explained above. This format is compatible with any Exceptional Model Mining framework implementation, but for this thesis we have created our own Python based implementation, which allows for more investigation and implementation options than an already existing implementation.

Case_ID	register request	examine thoroughly	check ticket	decide	reject request	examine casually	pay compensation	reinitiate request	Pete	Sue	Mike	Sara	Sean	Ellen
1	2	2	2	2	2	0	0	0	4	2	2	2	0	0
2	2	0	2	2	0	2	2	0	0	0	4	2	2	2
3	2	2	4	4	0	2	2	2	4	0	2	6	2	4
4	2	2	2	2	2	0	0	0	2	0	2	2	2	2
5	2	0	6	6	2	6	0	4	4	2	6	10	0	4
6	2	0	2	2	0	2	2	0	0	0	6	2	0	2

Case_ID	50	400	100	200	Fluxicon Nitro	start_date	end_date	duration	durHours	NoEvents	Average_Event_Duration	AEDIH
1	1	1	1	2	5	2010-12-30 10:02:00+00:00	2011-01-07 13:24:00+00:00	8 days 03:22:00	195.0	5	1 days 15:04:24	39.0
2	1	1	1	2	5	2010-12-30 10:32:00+00:00	2011-01-08 11:05:00+00:00	9 days 00:33:00	216.0	5	1 days 19:18:36	43.0
3	1	2	2	4	9	2010-12-30 13:32:00+00:00	2011-01-15 09:45:00+00:00	15 days 20:13:00	380.0	9	1 days 18:14:46.666666666	42.0
4	1	1	1	2	5	2011-01-06 14:02:00+00:00	2011-01-12 14:44:00+00:00	6 days 00:42:00	144.0	5	1 days 04:56:24	28.0
5	1	3	3	6	13	2011-01-06 08:02:00+00:00	2011-01-24 13:56:00+00:00	18 days 05:54:00	437.0	13	1 days 09:41:04.615384615	33.0
6	1	1	1	2	5	2011-01-06 14:02:00+00:00	2011-01-16 10:47:00+00:00	9 days 20:45:00	236.0	5	1 days 23:21:00	47.0

Figure 6: An example of a Case Attribute table

6 Experimental results

Now that it is possible to convert an event log to be usable with an Exceptional Model Mining framework implementation, we need to validate whether it is possible to obtain subgroups that contain traces which show divergent behaviour. For this, an actual event log is needed. For this thesis, we used the *running-examples* event log [2]. This event log describes a simple process where requests for compensation are registered and reviewed and it is decided whether the compensation is paid or the request is denied. This process can be seen as petri net in Figure 2 and in log format in Figure 1. The original log contains only six traces, which would result in a dataset with only six entries when transformed to the case attribute format which is compatible with Exceptional Model Mining.

Research on a dataset containing only six entries would never lead to substantial results. For this reason, we use the function *tree_generator* found in the *PM4PY* library [5] to generate the corresponding process tree. Using this process tree, it is possible to create an arbitrary number of simple traces using function *generate_log*, which only contain events. These traces are of arbitrary length, i.e., if a loop exists, this loop can be taken any number of times, and satisfy the model. The other attributes are stored as sets per event from the original log and are uniformly at random assigned to the corresponding events in the new traces. This way, a complete event log of a chosen length can be generated, which matches the trace, events and attributes structure of the original log. The only issue with this is that these events all follow each other with only one second apart.
To solve this issue, the average duration and start time for all events have been calculated from the original log, and this data has been used to edit the time of the generated cases.

After these operations have been performed, a correct event log can be generated which has a specified number of traces, which all correspond to the original model. We can now use this log to generate a case attributes table, from which subgroups can be generated. These subgroups can then be measured using some quality measure. Some possibilities for these measures will be discussed in Section 6.1.1.

6.1 Experimental setup, how did we perform our experiment?

The Exceptional Model Mining framework can discover subgroups that contain diverging behaviour. To do this it creates a model based on the user-specified target attributes. These generated models are based on the selected quality measure. For this thesis, we used *linear regression, Euclidean distance* and the *z-score* quality measures. These are explained in depth in Section 6.1.1. We also adapted the dataset, such that it contains traces that show deviating

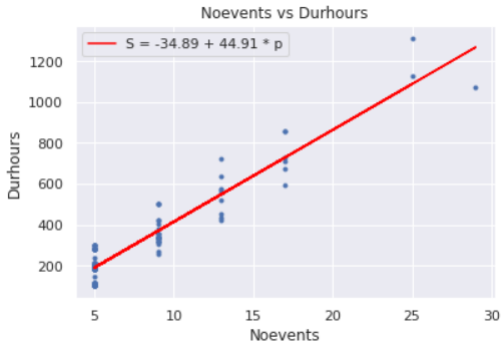


Figure 7: Plot of the entire dataset, with duration and number of events on the axis

behaviour. These traces are all modified to have a set of attributes, such that it can be discovered using Exceptional Model Mining. These modifications are described in detail in Section 6.1.2.

6.1.1 Which quality measures were used?

When subgroups have been generated, their divergence needs to be measured against either the complement or the entire dataset. For this a quality measure is used. Some quality measures have already been mentioned in Section 2.2.4.

First, we investigated the **regression model** [31] with two variables, since this can easily be visualized and thus is easier to understand. We selected the completion time and the number of events as our two variables, since saving time is usually one of the most desired gains when using Process Mining. Next, we plotted the entire dataset. As can be seen in Figure 7, the duration increases as the number of events increases. This behaviour is expected, when more events are executed, the total duration in hours increases.
Second, we investigated the **Euclidean distance** [30]. This method cannot be visualized when we are investigating more than 3 dimensions, but the idea behind it is easy to understand. For every target attribute in the subgroup, the average value is calculated. Next, the average for every target attribute in either the complement or entire dataset is calculated. Finally, the Euclidean distance is calculated. This distance is then used as quality measure.
Last, we investigated the **z-score** quality measure [21]. The standardized z-score of a subgroup with one target-attribute is defined by

$$\varphi_z = \frac{\mu - \mu_0}{\sigma_0/\sqrt{n}} = \frac{\sqrt{n}(\mu - \mu_0)}{\sigma_0}$$

where μ is the mean of the subgroup, μ_0 the mean of the entire dataset, σ_0 is the standard deviation of the entire dataset, and n is the size of the subgroup.

The standardized z-score measures how far the mean of the subgroup is away from the mean of the dataset D in terms of standard deviations. When using more than one target attribute, we need to adapt the formula slightly, we then end up with the following, adapted, z-score:

$$\varphi_z = \sum_{l=1}^{m} \frac{\sqrt{n}(\mu_l - \mu_{0l})}{\sigma_{0l}} = \sqrt{n} \sum_{l=1}^{m} \frac{(\mu_l - \mu_{0l})}{\sigma_{0l}}$$

The standardized z-score measures how far the mean of the subgroup is away from the mean of the dataset D in terms of standard deviations. This adapted z-score measures how far the mean of the target attributes is away from the mean of target attributes of the dataset D in terms of standard deviations.

6.1.2 How was the dataset modified?

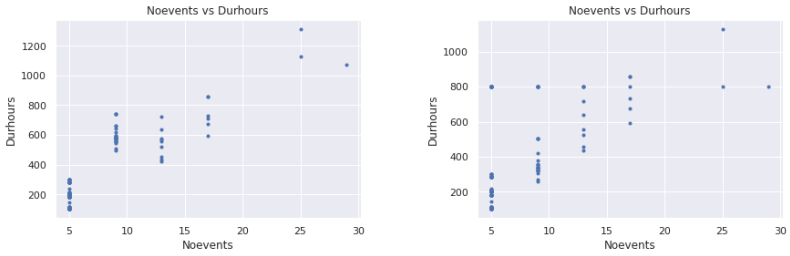
As of now, the dataset does not show strange behaviour, so we modified the dataset to show divergent behaviour. We created a department attribute, by determining for each event, which resource executed this event. We then created departments, such that every resource only exists in one department. Finally, for each event, we checked which resource executed it and added the correct department attribute. The reason for this is that we can now edit the event attributes for a certain department, such that this department will be the cause for divergent behaviour. This addition was made, because a department attribute covers more events, thus making it easier to discover. Now we can try different modifications and check whether Exceptional Model Mining is able to discover these subgroups using the quality measures mentioned in Section 6.1.1.

Alter the execution time of events executed by one department. This seems the most obvious option, increasing the execution time increases the total duration of the trace. This attribute is stored, so we can easily verify whether this can be detected. This can be done in multiple ways. We decided to try two methods. (1.) We take all traces of a specific number of events, assign them all the same department D3, and increase the execution time of these events. (2.) We take a random sample from the dataset, assign all traces the same department D3, and alter the execution time of these events.

When we alter the dataset according to (1.), we end up with the data distribution shown in Figure 8a. We immediately see that traces of length 9 behave differently compared to the rest of the dataset.

When we alter the dataset according to (2.), we end up with the data distribution shown in Figure 8b. The traces that take 800 hours to complete are easily spotted, since their behaviour is different than the majority of the dataset.

Now Exceptional Model Mining will be used using the quality measures described above to check whether Exceptional Model Mining is able to find this exceptional behaviour. We use a refinement depth of three and a minimum



(a) The execution time for traces of length 9 is increased

(b) The execution time for a sample of traces is set to 800

Figure 8: The plots of the data set, after is has been adapted

subgroup size of 20. We will compare against both the entire dataset and the complement to see whether this makes a difference. The obtained results are explained in Section 6.2 and Section 6.3.

6.2 Experiment 1: Increase the execution time for traces of length nine.

We adapt the dataset by increasing the duration in hours for traces of length nine. Next, we assign department D3 and new employees to these traces, such that they can be discovered. We use *regression*, *Euclidean distance* and *z-score* as quality measures to see whether they found the divergent traces.

6.2.1 Regression

We start with the comparison against *the entire dataset*. The descriptions are used to prune the dataset and the corresponding regression coefficient is calculated. When the slope is a vertical line, we assume it has has value ∞ for now. The quality measure calculates the difference between the two slopes. This value determines the exceptionality of the subgroup. Since traces of length 9 are edited, we expect to see subgroups where department D3 is involved. The slope of these subgroups will be ∞, since they all reside on the same x-axis value. What such a subgroup looks like can be seen in Figure 9. A subgroup contains traces, stored by index (Case_ID), with attribute-value pairs. This particular subgroup does not contain the desired interesting behaviour, as department *D3* is not involved in any of the cases. The description has the best score using linear regression, which we now know to be incorrect. The top-6 subgroups that were discovered can be seen in Figure 10.

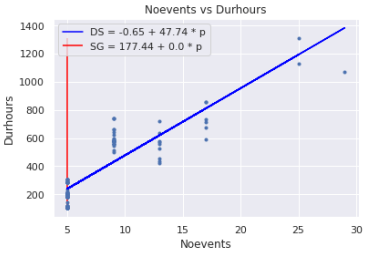
We quickly observe from the figures that we indeed end up with a vertical line. However, the number of events does not correspond to the expected result of nine. This can easily be explained, since the created subgroups all cover

Case_ID	register request	examine thoroughly	check ticket	decide	reject request	examine casually	pay compensation	reinitiate request	Pete	Sue	Mike	Sara	Sean	Ellen	Daniel	Martha	Erik	Chris	50	400	100	200	D0	D1	D2	D3	durHours	NoEvents	AEDIH
1	1	1	1	1	1	0	0	0	2	1	1	1	0	0	0	0	0	0	1	1	1	2	3	1	1	0	195.0	5	39.0
4	1	1	1	1	1	0	0	0	1	0	1	1	1	1	0	0	0	0	1	1	1	2	3	1	1	0	144.0	5	28.0
10	1	1	1	1	0	0	1	0	0	0	1	1	1	2	0	0	0	0	1	1	1	2	3	1	1	0	201.0	5	40.0
12	1	1	1	1	1	0	0	0	1	1	1	1	0	1	0	0	0	0	1	1	1	2	3	1	1	0	101.0	5	20.0
17	1	1	1	1	1	0	0	0	1	0	1	1	1	1	0	0	0	0	1	1	1	2	3	1	1	0	101.0	5	20.0
18	1	1	1	1	1	0	0	0	1	0	1	1	1	1	0	0	0	0	1	1	1	2	3	1	1	0	202.0	5	40.0
21	1	1	1	1	0	0	1	0	0	0	1	1	1	2	0	0	0	0	1	1	1	2	3	1	1	0	302.0	5	60.0
25	1	1	1	1	0	0	1	0	0	0	1	1	1	2	0	0	0	0	1	1	1	2	3	1	1	0	201.0	5	40.0
34	1	1	1	1	1	0	0	0	0	0	1	1	1	2	0	0	0	0	1	1	1	2	3	1	1	0	101.0	5	20.0
36	1	1	1	1	1	0	0	0	3	0	0	1	1	0	0	0	0	0	1	1	1	2	3	1	1	0	101.0	5	20.0
38	1	1	1	1	1	0	0	0	2	1	0	1	0	1	0	0	0	0	1	1	1	2	3	1	1	0	101.0	5	20.0
48	1	1	1	1	0	0	1	0	1	1	2	1	0	0	0	0	0	0	1	1	1	2	3	1	1	0	201.0	5	40.0
49	1	1	1	1	1	0	0	0	0	0	1	1	1	2	0	0	0	0	1	1	1	2	3	1	1	0	202.0	5	40.0
50	1	1	1	1	1	0	0	0	2	0	1	1	1	0	0	0	0	0	0	1	1	2	3	1	1	0	202.0	5	40.0
51	1	1	1	1	0	0	1	0	0	1	1	1	0	2	0	0	0	0	1	1	1	2	3	1	1	0	302.0	5	60.0
54	1	1	1	1	0	0	1	0	0	0	1	1	1	2	0	0	0	0	1	1	1	2	3	1	1	0	201.0	5	40.0
56	1	1	1	1	1	0	0	0	1	1	2	1	0	0	0	0	0	0	1	1	1	2	3	1	1	0	101.0	5	20.0
57	1	1	1	1	0	0	1	0	2	0	1	1	1	0	0	0	0	0	1	1	1	2	3	1	1	0	201.0	5	40.0
59	1	1	1	1	1	0	0	0	0	1	3	1	0	0	0	0	0	0	1	1	1	2	3	1	1	0	101.0	5	20.0
64	1	1	1	1	0	0	1	0	2	0	0	1	1	1	0	0	0	0	1	1	1	2	3	1	1	0	201.0	5	40.0
66	1	1	1	1	1	0	0	0	1	1	2	1	0	0	0	0	0	0	1	1	1	2	3	1	1	0	101.0	5	20.0
68	1	1	1	1	0	0	1	0	0	0	1	3	1	0	0	0	0	0	1	1	1	2	3	1	1	0	201.0	5	40.0
69	1	1	1	1	0	0	1	0	0	0	1	1	1	2	0	0	0	0	1	1	1	2	3	1	1	0	201.0	5	40.0
72	1	1	1	1	0	0	1	0	1	1	0	1	0	2	0	0	0	0	1	1	1	2	3	1	1	0	201.0	5	40.0
77	1	1	1	1	1	0	0	0	2	0	1	1	1	0	0	0	0	0	1	1	1	2	3	1	1	0	101.0	5	20.0
79	1	1	1	1	0	0	1	0	1	0	1	1	1	1	0	0	0	0	1	1	1	2	3	1	1	0	302.0	5	60.0
81	1	1	1	1	1	0	0	0	3	0	0	1	1	0	0	0	0	0	1	1	1	2	3	1	1	0	202.0	5	40.0
86	1	1	1	1	0	0	1	0	1	0	2	1	1	0	0	0	0	0	1	1	1	2	3	1	1	0	302.0	5	60.0
91	1	1	1	1	1	0	0	0	1	0	1	1	1	1	0	0	0	0	1	1	1	2	3	1	1	0	101.0	5	20.0
97	1	1	1	1	1	0	0	0	0	1	2	1	0	1	0	0	0	0	1	1	1	2	3	1	1	0	101.0	5	20.0
100	1	1	1	1	1	0	0	0	1	0	2	1	1	0	0	0	0	0	1	1	0	2	3	1	1	0	101.0	5	20.0
105	1	1	1	1	0	0	1	0	0	0	1	1	1	2	0	0	0	0	1	1	1	2	3	1	1	0	302.0	5	60.0

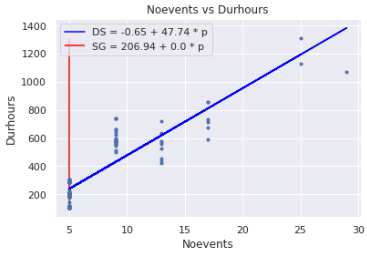
Figure 9: The resulting subgroup of query *examine thoroughly == 1 and register request == 1 and reinitiate request == 0*

traces which have the same number of events, this results in a line with slope ∞. Therefore, for the slope to be ∞, it does not matter how many events are in a trace, as long as all entries have the same number of events.

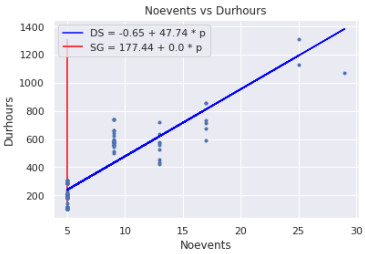
Next we compare it against the *complement of the dataset*. We expect to see the same results, since there will still be subgroups in which the traces all have the same number of events. These traces do not need to have length 9. We ran the discovery process again, and the top 6 results can be seen in Figure 11. As expected, the discovered subgroups all have a vertical line with a slope coefficient of ∞. They all cover traces of length 5, which is not the desired length of 9 events. A possible solution for this problem could be to ignore the traces with slope ∞. However, that solution would not work for this particular divergence in the data, since the desired subgroups also all have the same number of events. These would then also be ignored, and thus never show up.



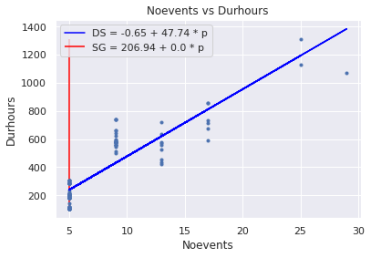
(a) ‘examine thoroughly‘ == 1 and ‘register request‘ == 1 and ‘reinitiate request‘ == 0



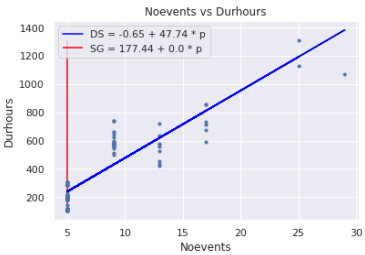
(b) ‘examine thoroughly‘ == 0 and ‘register request‘ == 1 and ‘reinitiate request‘ == 0



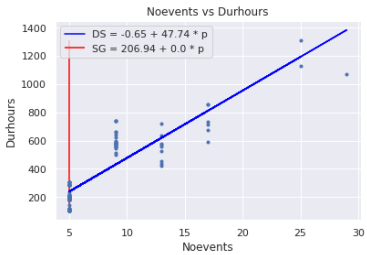
(c) ‘examine thoroughly‘ == 1 and ‘examine casually‘ == 0 and ‘reinitiate request‘ == 0



(d) ‘examine thoroughly‘ == 0 and ‘examine casually‘ == 1 and ‘reinitiate request‘ == 0

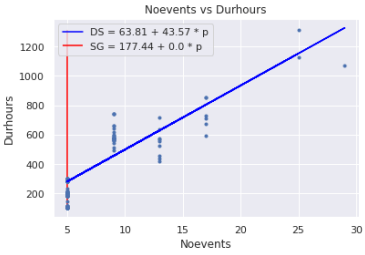


(e) ‘examine thoroughly‘ == 1 and ‘register request‘ == 1 and ‘examine casually‘ == 0

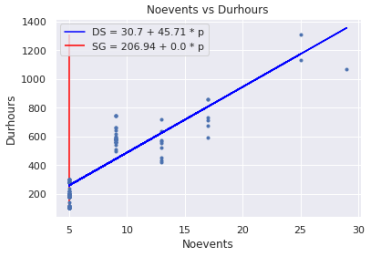


(f) ‘examine thoroughly‘ == 0 and ‘register request‘ == 1 and ‘examine casually‘ == 1

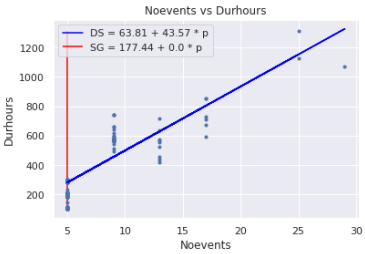
Figure 10: The top six discovered subgroups using regression, compared to the entire dataset.



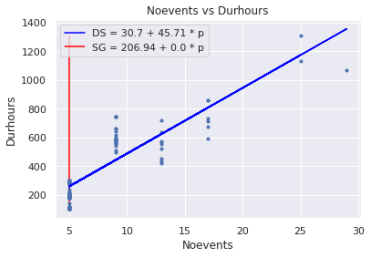
(a) ‘examine thoroughly‘ == 1 and ‘register request‘ == 1 and ‘reinitiate request‘ == 0



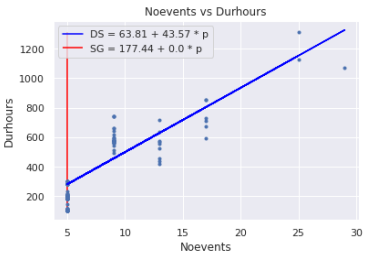
(b) ‘examine thoroughly‘ == 0 and ‘register request‘ == 1 and ‘reinitiate request‘ == 0



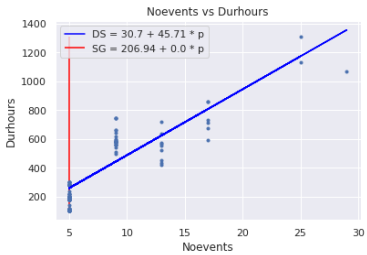
(c) ‘examine thoroughly‘ == 1 and ‘examine casually‘ == 0 and ‘reinitiate request‘ == 0



(d) ‘examine thoroughly‘ == 0 and ‘examine casually‘ == 1 and ‘reinitiate request‘ == 0



(e) ‘examine thoroughly‘ == 1 and ‘register request‘ == 1 and ‘examine casually‘ == 0

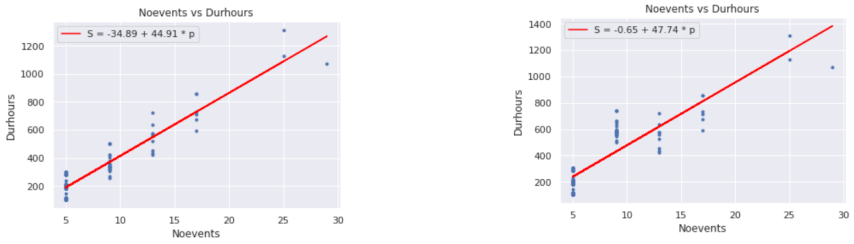


(f) ‘examine thoroughly‘ == 0 and ‘register request‘ == 1 and ‘examine casually‘ == 1

Figure 11: The top six discovered subgroups using regression, compared against the complement of the dataset.

6.2.2 Euclidean distance

We again start with the comparison against *the entire dataset*. We again use the six best descriptions to prune and discover the corresponding subgroups. For this discovery, we will use the Euclidean distance. However, since we only consider two attributes, we will still plot the top 6 subgroups. This way we can easily visualize and interpret the results. The top 6 descriptions are shown as plots in Figure 13. We see that these subgroups do not contain traces that have



(a) The original dataset

(b) The modified dataset

Figure 12: The difference in regression plot, after increasing the duration of traces of length nine

the desired length of 9 events. This is caused by the fact that for the Euclidean distance, we calculate the mean of the target attributes for both the subgroup and the entire dataset, and calculate the respective distance. First, we notice that the traces with length 5 have the lowest mean when looking at the number of events, and thus lie farthest away from mean of the dataset. Secondly, when we look at Figure 12, we observe that the distance between the plotted line and the shortest traces of length five has increased, compared to the original dataset. The Euclidean distance for the first subgroup with description *'reject request' == 1 and '50' == 1 and 'D0' == 3* is 237.4, whereas the distance for *'NoEvents' == 9* is 223.6. We see that the first value of the equation in the figures is the same for the first two graphs. This is an indication that the descriptions are part of a longer description, which is not discovered, since the refinement depth was reached before this description could be discovered. The corresponding descriptions are *'D0' == 3 and '50' == 1 and 'pay compensation' == 0* and *'D0' == 3 and '50' == 1 and 'reject request' == 1*. When we look at Figure 2, we indeed see that these describe the same subgroup, since the choice between either *reject request*, *pay compensation* and *reinitiate request* has to be made, in this case, the request was rejected. When combining both descriptions, we would get *'D0' == 3 and '50' == 1 and 'reject request' == 1 and 'pay compensation' == 0* which has the same score of 237.4 as mentioned earlier.

Next, we investigate the results of the best six subgroups when compared to *the complement of the dataset*. The subgroups can again be seen as plots in Figure 14. We again have subgroups with 5 events. This time, it is observed that all plots have the same first value in the equations. Thus, when applying the same reasoning as before, we expect to see descriptions which are all subsets of one large description, describing the same subgroup. We obtained the following descriptions and their corresponding scores:

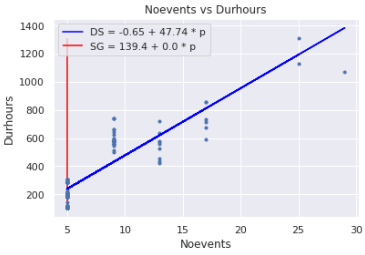
1. *'reinitiate request' == 0 and 'check ticket' == 1 and 'register request' == 1*, 455.66

2. *'reinitiate request' == 0 and 'Martha' == 0 and 'register request' == 1*,

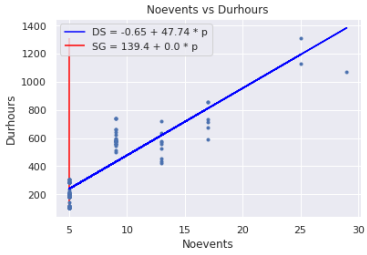
455.66

3. *'reinitiate request' == 0 and 'register request' == 1 and 'decide' == 1*, 455.66
4. *'reinitiate request' == 0 and 'register request' == 1 and 'Daniel' == 0*, 455.66
5. *'reinitiate request' == 0 and 'Chris' == 0 and 'register request' == 1*, 455.66
6. *'reinitiate request' == 0 and 'Erik' == 0 and 'register request' == 1*, 455.66

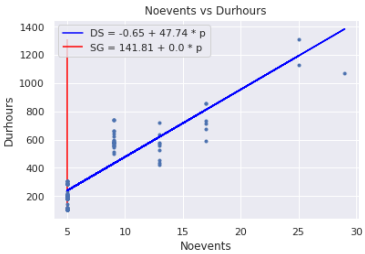
These all have two attributes in common, and all result in the same score, so we conclude that these all describe the same subgroup. However, it is still not the subgroup which we wanted to see, that describes traces of length nine.



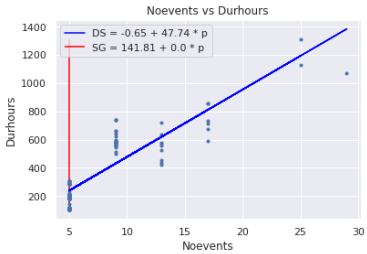
(a) ‘D0‘ == 3 and ‘50‘ == 1 and ‘pay compensation‘ == 0



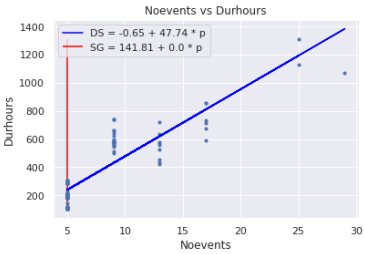
(b) ‘D0‘ == 3 and ‘50‘ == 1 and ‘reject request‘ == 1



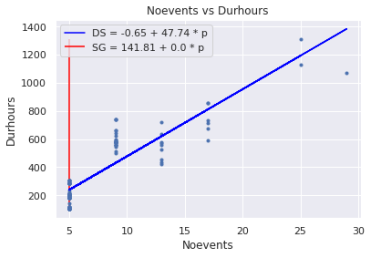
(c) ‘D0‘ == 3 and ‘reinitiate request‘ == 0 and ‘pay compensation‘ == 0



(d) ‘reinitiate request‘ == 0 and ‘D1‘ == 1 and ‘pay compensation‘ == 0

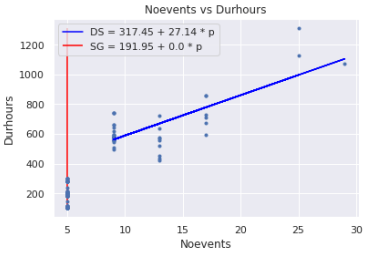


(e) ‘register request‘ == 1 and ‘D0‘ == 3 and ‘pay compensation‘ == 0

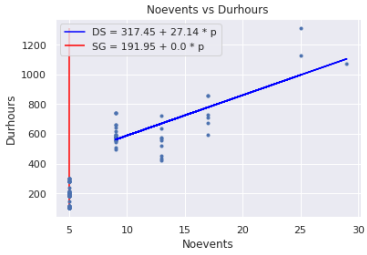


(f) ‘D0‘ == 3 and ‘reinitiate request‘ == 0 and ‘reject request‘ == 1

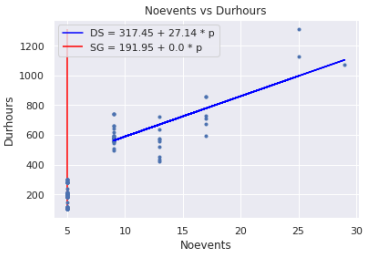
Figure 13: The top six discovered subgroups using Euclidean distance, compared against the entire dataset.



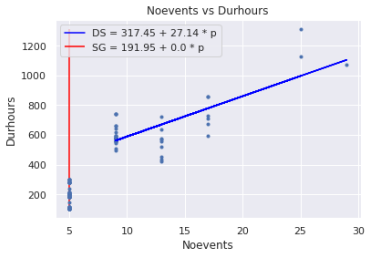
(a) ‘reinitiate request‘ == 0 and ‘check ticket‘ == 1 and ‘register request‘ == 1



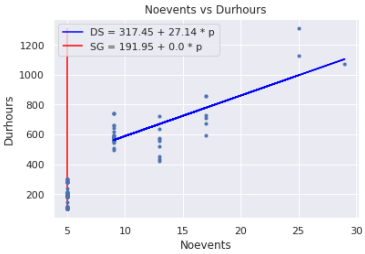
(b) ‘reinitiate request‘ == 0 and ‘Martha‘ == 0 and ‘register request‘ == 1



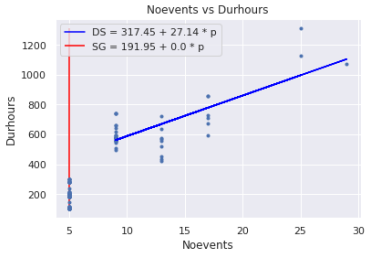
(c) ‘reinitiate request‘ == 0 and ‘register request‘ == 1 and ‘decide‘ == 1



(d) ‘reinitiate request‘ == 0 and ‘register request‘ == 1 and ‘Daniel‘ == 0



(e) ‘reinitiate request‘ == 0 and ‘Chris‘ == 0 and ‘register request‘ == 1



(f) ‘reinitiate request‘ == 0 and ‘Erik‘ == 0 and ‘register request‘ == 1

Figure 14: The top six discovered subgroups using Euclidean distance, compared against the complement of the dataset.

6.2.3 Z-score

Finally, we look at the results obtained from the z-score quality measure. We first start with the comparison against *the entire dataset*. For readability, it is again shown in a plot. The top 6 scoring descriptions and their corresponding subgroups are shown in Figure 16. We observe that the descriptions describe traces which have length nine. These are the traces that we modified and thus are interested in. We again notice that the first value in the equation is the same for all subgroups, so we again suspect that the descriptions are subsets of

a larger description. The subgroups have the following descriptions and score.

1. *'register request' == 1 and 'check ticket' == 2 and 'reinitiate request' == 1*, 5.54
2. *'register request' == 1 and 'reinitiate request' == 1 and 'decide' == 2*, 5.54
3. *'register request' == 1 and 'Ellen' == 0 and 'reinitiate request' == 1*, 5.54
4. *'Sean' == 0 and 'register request' == 1 and 'reinitiate request' == 1*, 5.54
5. *'register request' == 1 and 'Sara' == 0 and 'reinitiate request' == 1*, 5.54
6. *'register request' == 1 and 'Mike' == 0 and 'reinitiate request' == 1*, 5.54

These all have the same score and have at least two attributes in common. If we were to combine these descriptions we would get one large description which would contain all attributes. *'register request' == 1 and 'check ticket' == 2 and 'reinitiate request' == 1 and 'decide' == 2 and 'Ellen' == 0 and 'Sean' == 0 and 'Sara' == 0 and 'Mike' == 0*, which has score 5.54. To be completely sure, the description *'D3' == 9* was checked, and this also results in score 5.54. We are now completely convinced that using this quality measure, we found the correct subgroup.

Finally, we make the comparison against *the complement of the dataset*, which resulted in the subgroups found in Figure 17. These subgroups show different behaviour from what we have seen so far. The plot corresponding to the subgroups has a slope only slightly different from the entire dataset. This is an indication that the subgroup consists traces as if they were taken as a random sample from the dataset. When we look at Figure 15, which is the plot for the first subgroup, we see that this is indeed the case. This can be explained due to the fact that in order for the z-score to be as high as possible, the difference between μ and μ_0 should be maximized, while keeping the standard deviation σ_0 as low as possible. The easiest way to achieve this is to select outliers, since these have a higher mean, and result in a decrease of the standard deviation. However, in this case this is not possible, since the subgroups would not be large enough, if only outliers were selected. We noticed that the plot of the subgroup was higher than the plot of the complement. This means that the difference in the mean is maximized, while the standard deviation is kept to a minimum. The score for these subgroups is 7.51, while the score for *'D3' == 9* is 7.12. Descriptions describing the same subgroup did score well, but the pseudo random sample scored slightly better.

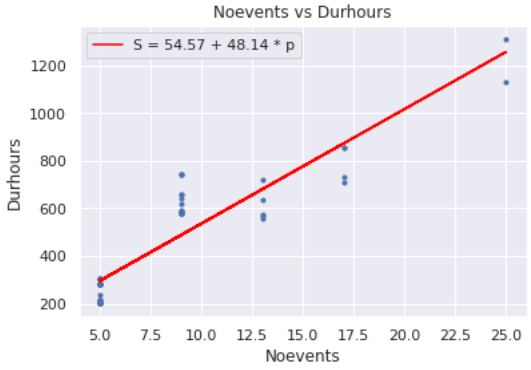
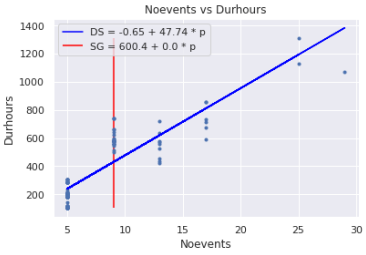
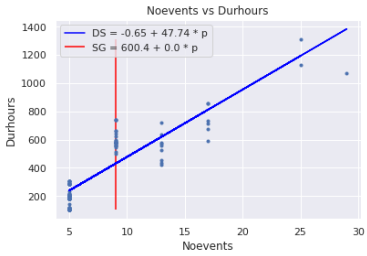


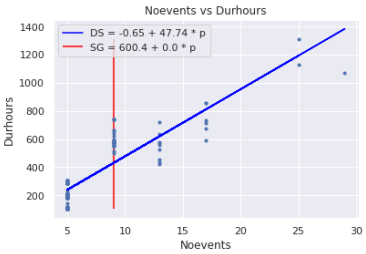
Figure 15: The plot for the first subgroup obtained from the comparison against the complement of the dataset



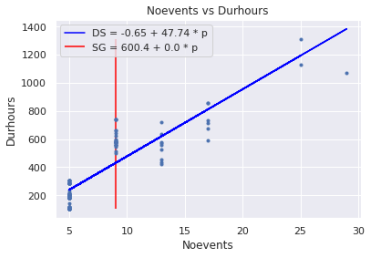
(a) ‘register request‘ == 1 and ‘check ticket‘ == 2 and ‘reinitiate request‘ == 1



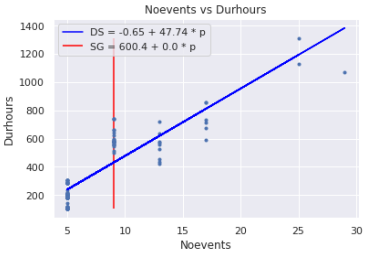
(b) ‘register request‘ == 1 and ‘reinitiate request‘ == 1 and ‘decide‘ == 2



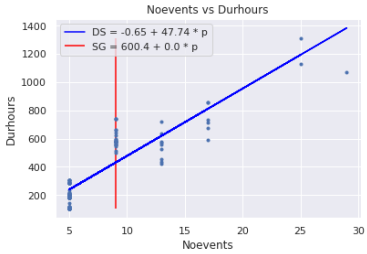
(c) ‘register request‘ == 1 and ‘Ellen‘ == 0 and ‘reinitiate request‘ == 1



(d) ‘Sean‘ == 0 and ‘register request‘ == 1 and ‘reinitiate request‘ == 1

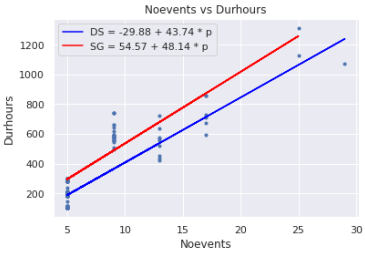


(e) ‘register request‘ == 1 and ‘Sara‘ == 0 and ‘reinitiate request‘ == 1

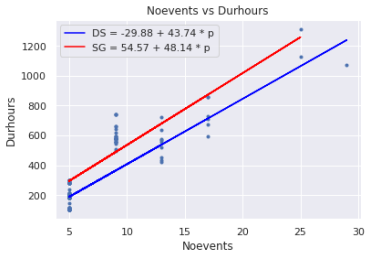


(f) ‘register request‘ == 1 and ‘Mike‘ == 0 and ‘reinitiate request‘ == 1

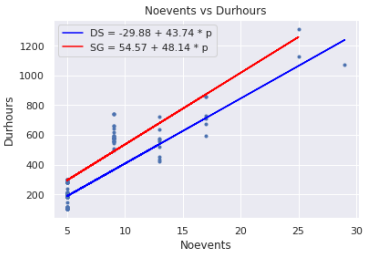
Figure 16: The top six discovered subgroups using z-score, compared against the entire dataset.



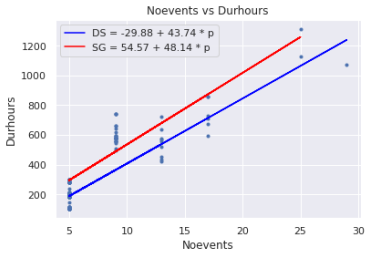
(a) ‘register request‘ == 1 and ‘pay compensation‘ == 1 and ‘reject request‘ == 0



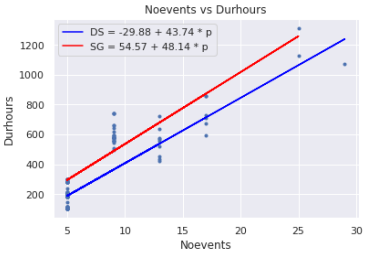
(b) ‘register request‘ == 1 and ‘50‘ == 1 and ‘pay compensation‘ == 1



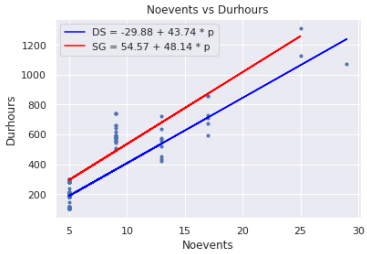
(c) ‘register request‘ == 1 and ‘50‘ == 1 and ‘reject request‘ == 0



(d) ‘50‘ == 1 and ‘pay compensation‘ == 1 and ‘reject request‘ == 0



(e) ‘register request‘ == 1 and ‘pay compensation‘ == 1



(f) ‘register request‘ == 1 and ‘reject request‘ == 0

Figure 17: The top six discovered subgroups using z-score, compared against the complement of the dataset.

6.3 Experiment 2: Alter the total execution time for a random sample of the dataset.

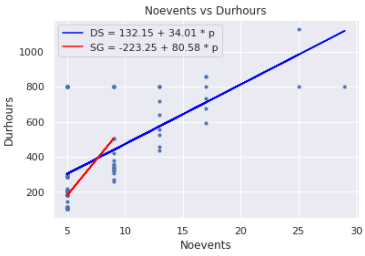
Using the *random* function in Python, we uniformly at random select cases using their identifier. Next we alter the execution time to be the same for all of the traces in the sample. We also assign department D3 and new employees to the traces in the sample, such that we can verify whether Exceptional Model Mining is able to discover these traces. A plot of the dataset can be seen in Figure 8b. We again look at the three quality measures *regression*, *Euclidean distance* and

z-score and compare the subgroups against the entire and the complement of the dataset.

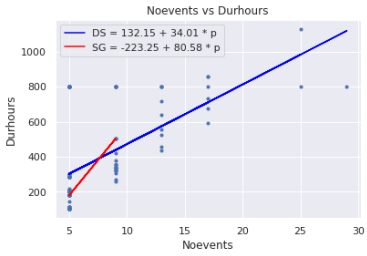
6.3.1 Regression

We again start with the top-6 scoring subgroups when we use regression and compare the subgroups against *the entire dataset*. We made a small adaption to the implementation however. In the previous Section we saw that subgroups with a vertical line scored best, due to their slope of ∞. Since we were looking for a vertical line as well, this was a desired property. However, in the current section we do not want subgroups with a vertical slope, because the desired subgroup will have a horizontal slope. So, if a subgroup describes traces of the same length, we give it score 0, to prevent it from showing up. The top-6 subgroups can be seen in Figure 18. The adaption to the implementation works, since we do not see subgroups which describe traces of one length. We do however, still not see the desired results. This is because the difference in slope for the discovered subgroup and the dataset is larger than the difference between the desired subgroup and the dataset. This happens due to the fact that the only requirement for the subgroups is the size. If the requirements would be more constraining, one could include the restriction that subgroups would need to contain traces of at least three different numbers of events. This way, only subgroups which have a bigger spread over the entire dataset are selected, and thus decreasing the slope of the subgroup. However, this is such a precise restriction that we did not use it. We also see that the slope for the top-6 subgroups is the same, so they again are descriptions that describe the same subgroup.

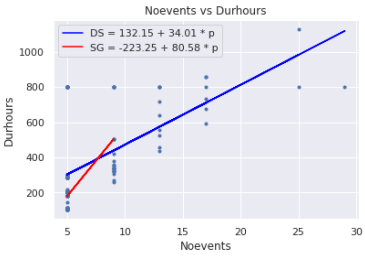
Next we compare the subgroups against *the complement of the dataset*. The results can be seen in Figure 19. We see the exact same results as when compared against the entire dataset.



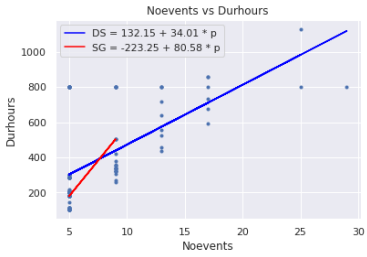
(a) ‘D1‘ == 1 and ‘Pete‘ == 1 and ‘register request‘ == 1



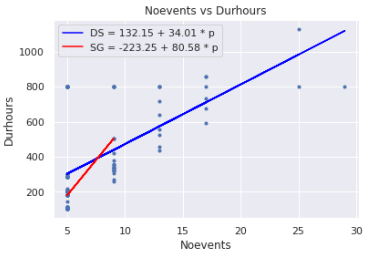
(b) ‘D1‘ == 1 and ‘Pete‘ == 1 and ‘Martha‘ == 0



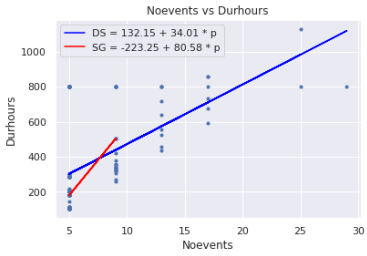
(c) ‘D1‘ == 1 and ‘Pete‘ == 1 and ‘Daniel‘ == 0



(d) ‘D1‘ == 1 and ‘Pete‘ == 1 and ‘Chris‘ == 0

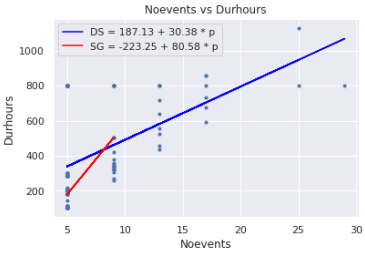


(e) ‘D1‘ == 1 and ‘Pete‘ == 1 and ‘Erik‘ == 0

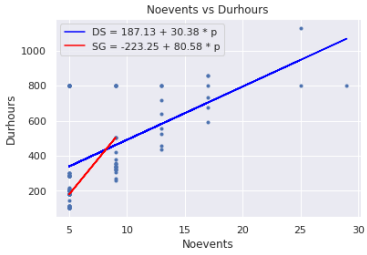


(f) ‘D1‘ == 1 and ‘50‘ == 1 and ‘Pete‘ == 1

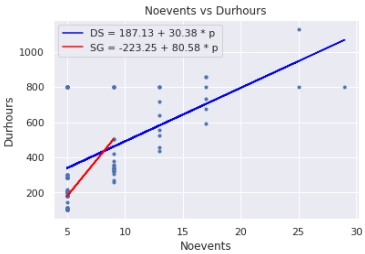
Figure 18: The top six discovered subgroups using regression, compared against the entire dataset.



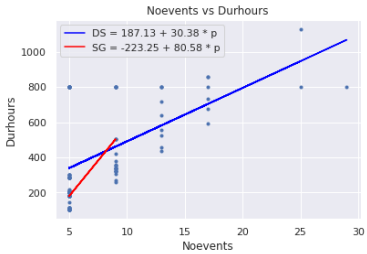
(a) ‘D1‘ == 1 and ‘Pete‘ == 1 and ‘register request‘ == 1



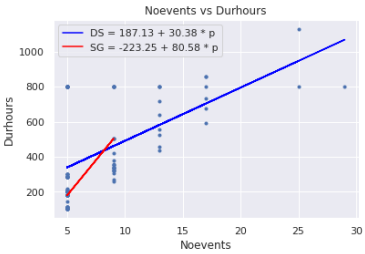
(b) ‘D1‘ == 1 and ‘Pete‘ == 1 and ‘Martha‘ == 0



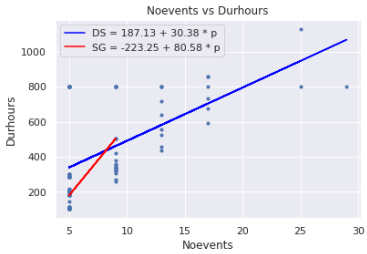
(c) ‘D1‘ == 1 and ‘Pete‘ == 1 and ‘Daniel‘ == 0



(d) ‘D1‘ == 1 and ‘Pete‘ == 1 and ‘Chris‘ == 0



(e) ‘D1‘ == 1 and ‘Pete‘ == 1 and ‘Erik‘ == 0



(f) ‘D1‘ == 1 and ‘50‘ == 1 and ‘Pete‘ == 1

Figure 19: The top six discovered subgroups using regression, compared against the complement of the dataset.

6.3.2 Euclidean distance

Next, we used Euclidean distance as quality measure to discover subgroups. The top-6 subgroups are plotted in Figure 20. The results are promising, we see the desired subgroup of traces with a duration of 800 hours. The corresponding descriptions and their corresponding score are:

1. *‘register request‘ == 1 and ‘Sara‘ == 0 and ‘Ellen‘ == 0*, 398.95
2. *‘Sean‘ == 0 and ‘register request‘ == 1 and ‘Sara‘ == 0*, 398.95
3. *‘register request‘ == 1 and ‘Sara‘ == 0 and ‘Mike‘ == 0*, 398.95

4. *'register request' == 1 and 'Sara' == 0 and 'Pete' == 0*, 398.95
5. *'register request' == 1 and 'D2' == 0 and 'Ellen' == 0*, 398.95
6. *'register request' == 1 and 'Sara' == 0 and 'Sue' == 0*, 398.95

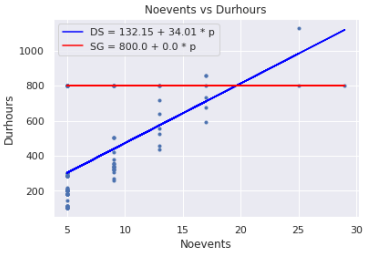
These can all be combined into one long description *'register request' == 1 and 'Sara' == 0 and 'Ellen' == 0 and 'Sean' == 0 and 'Mike' == 0 and 'Pete' == 0 and 'Sue' == 0*, which has the same score of 398.95. These descriptions are indirectly an indication that the correct department is chosen. Namely, all other employees in the company are not involved in the case. But to be 100% certain, we check the corresponding subgroup and see that all cases have a completion time of 800 hours.

For Euclidean distance, we also compare it against the complement of the dataset. The plots of the top-6 scoring subgroups can be seen in Figure 21. For the first five plots, something happens that has not happened before. The first five subgroups contain the data that is in the complement of the sixth one. They are opposites of each other. Therefore the first five descriptions describe subgroups that contain all traces except the traces that took 800 hours to complete, whereas the last description describes the desired traces with duration of 800 hours. The subgroups are opposites of each other, since they all have the same Euclidean distance of 497.51.

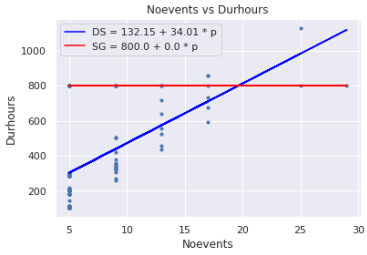
The descriptions and their corresponding scores that belong to the subgroups are the following:

1. *'register request' == 1 and 'Martha' == 0 and 'Daniel' == 0*, 497.51
2. *'Erik' == 0 and 'register request' == 1 and 'Martha' == 0*, 497.51
3. *'Erik' == 0 and 'register request' == 1 and 'Daniel' == 0*, 497.51
4. *'Chris' == 0 and 'Erik' == 0 and 'register request' == 1*, 497.51
5. *'register request' == 1 and 'Martha' == 0 and 'D3' == 0*, 497.51
6. *'register request' == 1 and 'Sara' == 0 and 'Ellen' == 0*, 497.51

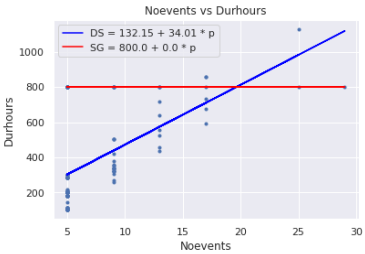
We can again combine the first five descriptions into one large description. We see an attribute *'D3' == 0*, which indeed confirms that this describes all traces except the traces with duration of 800 hours. For the sixth description, we checked the dataset and indeed notice that this describes traces with a duration of 800 hours.



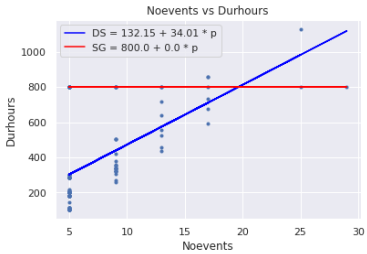
(a) ‘register request‘ == 1 and ‘Sara‘ == 0 and ‘Ellen‘ == 0



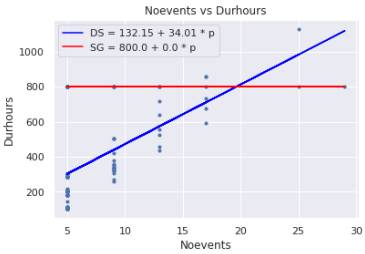
(b) ‘Sean‘ == 0 and ‘register request‘ == 1 and ‘Sara‘ == 0



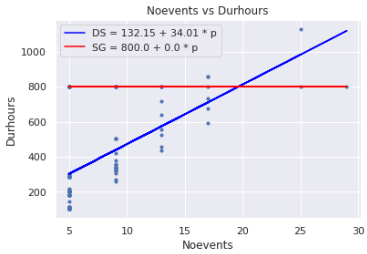
(c) ‘register request‘ == 1 and ‘Sara‘ == 0 and ‘Mike‘ == 0



(d) ‘register request‘ == 1 and ‘Sara‘ == 0 and ‘Pete‘ == 0

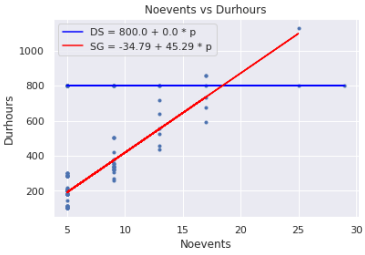


(e) ‘register request‘ == 1 and ‘D2‘ == 0 and ‘Ellen‘ == 0

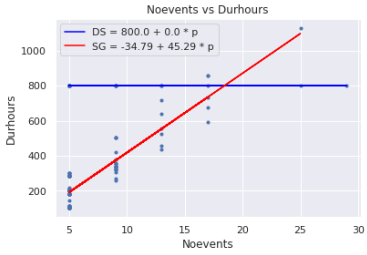


(f) ‘register request‘ == 1 and ‘Sara‘ == 0 and ‘Sue‘ == 0

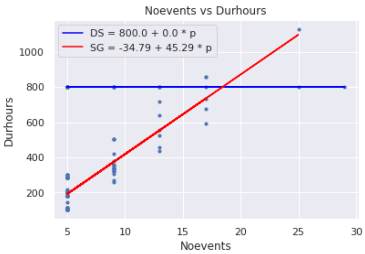
Figure 20: The top six discovered subgroups using Euclidean distance, compared against the entire dataset.



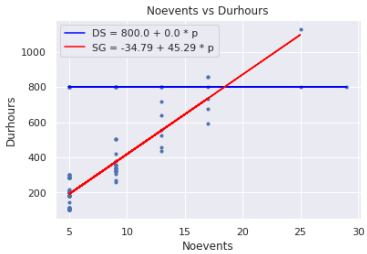
(a) ‘register request‘ == 1 and ‘Martha‘ == 0 and ‘Daniel‘ == 0



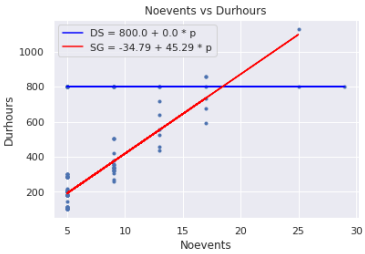
(b) ‘Erik‘ == 0 and ‘register request‘ == 1 and ‘Martha‘ == 0



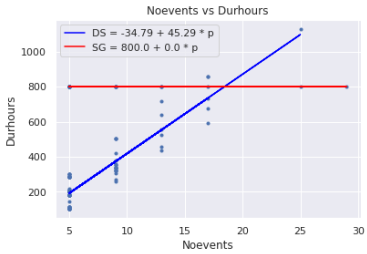
(c) ‘Erik‘ == 0 and ‘register request‘ == 1 and ‘Daniel‘ == 0



(d) ‘Chris‘ == 0 and ‘Erik‘ == 0 and ‘register request‘ == 1



(e) ‘register request‘ == 1 and ‘Sara‘ == 0 and ‘Ellen‘ == 0



(f) ‘register request‘ == 1 and ‘Martha‘ == 0 and ‘D3‘ == 0

Figure 21: The top six discovered subgroups using Euclidean distance, compared against the complement of the dataset.

6.3.3 Z-score

The z-score is the final quality measure we use for this dataset. We again start with the comparison against the entire dataset, and the top-6 subgroups are plotted in Figure 22. Again, we are able to discover the correct traces using the z-score quality measure. All subgroups have the same score of 8.74, which can be seen when we look at all descriptions and their corresponding score again.

1. *‘register request‘ == 1 and ‘Sara‘ == 0 and ‘Ellen‘ == 0* 8.74
2. *‘Sean‘ == 0 and ‘register request‘ == 1 and ‘Sara‘ == 0* 8.74

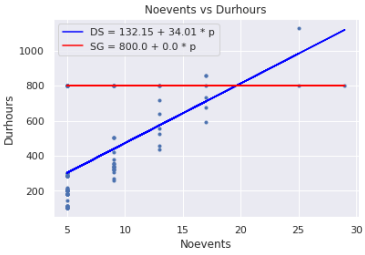
3. *'register request' == 1 and 'Sara' == 0 and 'Mike' == 0* 8.74
4. *'register request' == 1 and 'Sara' == 0 and 'Pete' == 0* 8.74
5. *'register request' == 1 and 'D2' == 0 and 'Ellen' == 0* 8.74
6. *'register request' == 1 and 'Sara' == 0 and 'Sue' == 0* 8.74

If all these descriptions were combined, we would end up with one description *'register request' == 1 and 'Sara' == 0 and 'Ellen' == 0 and 'Sean' == 0 and 'Mike' == 0 and 'Pete' == 0 and 'D2' == 0 and 'Sue' == 0*, which again results in a score of 8.74. This is no direct indication that it only contains traces with a duration of 800 hours, but as was the case with the previous z-score, the attributes rule out all other options. To be 100% sure, we checked the description *'D3' > 0* and indeed get the same score of 8.74.

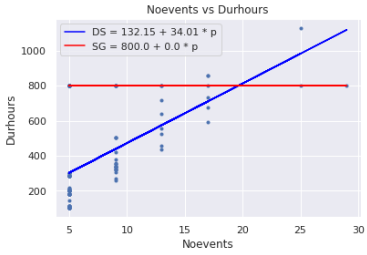
Finally, we use the z-score quality measure for creating subgroups that we compare against the complement of the dataset. The top-6 subgroups are plotted in Figure 23. Unlike with the previous dataset, this time, we are able to discover the correct traces. The following descriptions and their score represent the best subgroups.

1. *'register request' == 1 and 'Sara' == 0 and 'Ellen' == 0* 14.48
2. *'Sean' == 0 and 'register request' == 1 and 'Sara' == 0* 14.48
3. *'register request' == 1 and 'Sara' == 0 and 'Mike' == 0* 14.48
4. *'register request' == 1 and 'Sara' == 0 and 'Pete' == 0* 14.48
5. *'register request' == 1 and 'D2' == 0 and 'Ellen' == 0* 14.48
6. *'register request' == 1 and 'Sara' == 0 and 'Sue' == 0* 14.48

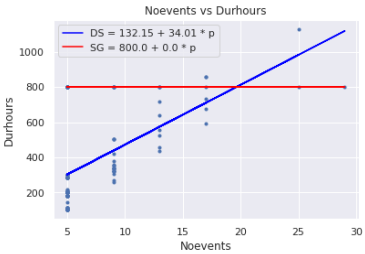
We see that the descriptions are the same as for the comparison against the entire dataset. So the same subgroups are found using this quality measure. This is different than with the previous dataset, where it was unable to find the correct subgroups.



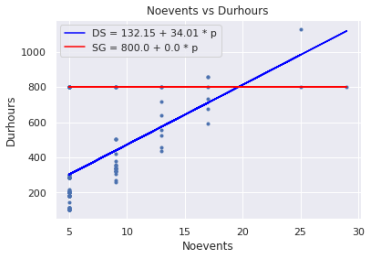
(a) ‘register request‘ == 1 and ‘Sara‘ == 0 and ‘Ellen‘ == 0



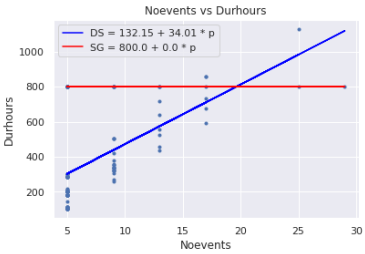
(b) ‘Sean‘ == 0 and ‘register request‘ == 1 and ‘Sara‘ == 0



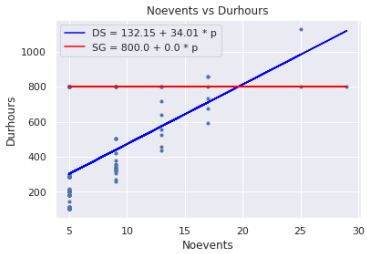
(c) ‘register request‘ == 1 and ‘Sara‘ == 0 and ‘Mike‘ == 0



(d) ‘register request‘ == 1 and ‘Sara‘ == 0 and ‘Pete‘ == 0

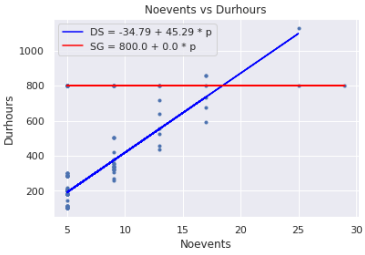


(e) ‘register request‘ == 1 and ‘D2‘ == 0 and ‘Ellen‘ == 0

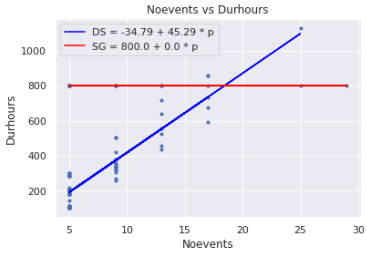


(f) ‘register request‘ == 1 and ‘Sara‘ == 0 and ‘Sue‘ == 0

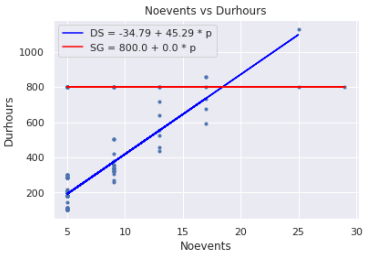
Figure 22: The top six discovered subgroups using z-score, compared against the entire dataset.



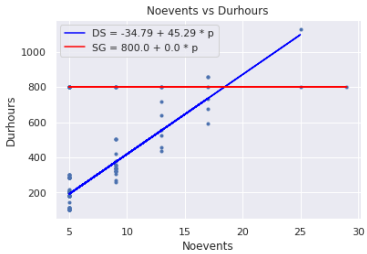
(a) ‘register request‘ == 1 and ‘Sara‘ == 0 and ‘Ellen‘ == 0



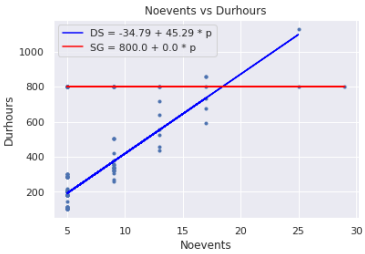
(b) ‘Sean‘ == 0 and ‘register request‘ == 1 and ‘Sara‘ == 0



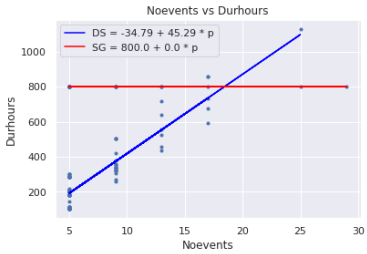
(c) ‘register request‘ == 1 and ‘Sara‘ == 0 and ‘Mike‘ == 0



(d) ‘register request‘ == 1 and ‘Sara‘ == 0 and ‘Pete‘ == 0



(e) ‘register request‘ == 1 and ‘D2‘ == 0 and ‘Ellen‘ == 0



(f) ‘register request‘ == 1 and ‘Sara‘ == 0 and ‘Sue‘ == 0

Figure 23: The top six discovered subgroups using z-score, compared against the complement of the dataset.

All three quality measures have been used in two modified datasets, to investigate whether they can be used to discover the desired subgroups. Of the three quality measures, the z-score quality measure performs best. We also observed that it is best to compare against the complete dataset, since this seems to give the most reliable results. Using this comparison, when using the z-score quality measure, we were able to discover the divergent subgroup in both cases. When using the entire dataset, we were also able to discover the desired subgroup using Euclidean distance for the second dataset. When comparing against the complement, we discovered the rest of the dataset as our subgroup, and the desired traces as the complement, which was the exact opposite of what we wanted.

This is explained by the fact that the Euclidean distance between two datasets is the same, and order does not matter. So when comparing a subgroup and its complement, the order in which these are used in the calculation does not make a difference. So the distance from subgroup to complement is the same as complement to subgroup. Thus, for future research, it is likely that the z-score quality measure will give the best results. It should however be noted that the choice of quality measure is up to the user, so using the z-score quality measure is nothing more than an advice.

Now that the results obtained using Exceptional Model Mining have been analysed from an Exceptional Model Mining point of view, it is time to analyse and validate these results using a Process Mining tool. Since we created an Exceptional Model Mining implementation in Python, we decided to use PM4PY [5] to analyse the original traces.

6.4 Downstream employment of results in Process Mining

u In the previous section, the obtained results were analysed from an Exceptional Model Mining point of view. For every dataset and quality measure, we investigated the obtained descriptions and determined whether they discovered the correct subgroup we had manually created. In this section, we will discuss how these results should be interpreted and used from a Process Mining point of view.

6.4.1 How should the user interpret the results of Exceptional Model Mining?

The discovered subgroups have a description. However, as seen in the previous section, it should be noted that these descriptions do not describe every attribute from the subgroup. It can be the case that a description describes the correct subgroup, but that it does not use the expected attributes to do so. We use the description to prune the dataset and look at the corresponding subgroup. We do this for both datasets from the previous section using the top description obtained using the z-score quality measure.

All traces with nine events have an increased execution time:
As we saw in Figure 8a, the execution time for the traces of length nine is increased noticeably. Using Exceptional Model Mining we were also able to discover these subgroups using the z-score quality measure. The best description was *'register request' == 1 and 'check ticket' == 2 and 'reinitiate request' == 1.* When we use this description to prune the dataset, we obtain the subgroup as shown in Figure 24. When analyzing this dataset, we notice that only department D3 is involved when executing these traces. In this case, this can thus be used as a trace attribute for pruning the original event log.

A sample of traces is modified such that it has an execution time of 800 hours: As seen in Figure 8b, the execution time of traces for a random sample of the data is modified such that they have an execution time of 800 hours. We again used the z-score quality measure, since this gave the best results. For pruning the dataset, we used the top description discovered using the z-score quality measure to describe cases with a duration of 800 hours, performed by department D3. This is *'register request' == 1 and 'Sara' == 0 and 'Ellen' == 0.* This does not directly indicate that department D3 is involved. The user can use the description to obtain the subgroup it describes, and check whether other attributes might be more useful for pruning the original event log. The corresponding dataset can be seen in Figure 25. When analyzing this dataset, we see that only department D3 is involved. This is for the corresponding traces a trace attribute, which can be used for pruning the Event Data.

Case_ID	register request	examine thoroughly	check ticket	decide	reject request	examine casually	pay compensation	reinitiate request	Pete	Sue	Mike	Sara	Sean	Ellen	Martha	Erik	Chris	Daniel	50	400	100	200	D0	D1	D2	D3	durHours	NoEvents	AEDIH
3	1	1	2	2	0	1	1	1	0	0	0	0	0	0	4	3	1	1	1	2	2	4	0	0	0	9	620.0	9	68.0
7	1	1	2	2	1	1	0	1	0	0	0	0	0	0	2	1	2	4	1	2	2	4	0	0	0	9	560.0	9	62.0
8	1	2	2	2	0	0	1	1	0	0	0	0	0	0	3	1	3	2	1	2	2	4	0	0	0	9	578.0	9	64.0
24	1	1	2	2	1	1	0	1	0	0	0	0	0	0	3	3	1	2	1	2	2	4	0	0	0	9	511.0	9	56.0
33	1	2	2	2	0	0	1	1	0	0	0	0	0	0	4	3	1	1	1	2	2	4	0	0	0	9	578.0	9	64.0
35	1	1	2	2	0	1	1	1	0	0	0	0	0	0	2	3	2	2	1	2	2	4	0	0	0	9	660.0	9	73.0
39	1	1	2	2	1	1	0	1	0	0	0	0	0	0	3	3	2	1	1	2	2	4	0	0	0	9	560.0	9	62.0
41	1	0	2	2	0	2	1	1	0	0	0	0	0	0	3	0	3	3	1	2	2	4	0	0	0	9	742.0	9	82.0
42	1	1	2	2	1	1	0	1	0	0	0	0	0	0	1	2	4	2	1	2	2	4	0	0	0	9	579.0	9	64.0
46	1	1	2	2	0	1	1	1	0	0	0	0	0	0	3	2	2	2	1	2	2	4	0	0	0	9	592.0	9	65.0
53	1	2	2	2	0	0	1	1	0	0	0	0	0	0	3	4	2	0	1	2	2	4	0	0	0	9	578.0	9	64.0
55	1	0	2	2	0	2	1	1	0	0	0	0	0	0	5	3	0	1	1	2	2	4	0	0	0	9	742.0	9	82.0
61	1	1	2	2	0	1	1	1	0	0	0	0	0	0	2	1	4	2	1	2	2	4	0	0	0	9	660.0	9	73.0
62	1	2	2	2	0	0	1	1	0	0	0	0	0	0	3	4	0	2	1	2	2	4	0	0	0	9	578.0	9	64.0
70	1	0	2	2	0	2	1	1	0	0	0	0	0	0	1	2	2	4	1	2	2	4	0	0	0	9	742.0	9	82.0
71	1	1	2	2	0	1	1	1	0	0	0	0	0	0	2	4	1	2	1	2	2	4	0	0	0	9	592.0	9	65.0
75	1	1	2	2	1	1	0	1	0	0	0	0	0	0	3	1	4	1	1	2	2	4	0	0	0	9	579.0	9	64.0
76	1	2	2	2	1	0	0	1	0	0	0	0	0	0	1	1	3	4	1	2	2	4	0	0	0	9	597.0	9	66.0
78	1	1	2	2	1	1	0	1	0	0	0	0	0	0	0	1	4	4	1	2	2	4	0	0	0	9	544.0	9	60.0
83	1	1	2	2	1	1	0	1	0	0	0	0	0	0	2	2	1	4	1	2	2	4	0	0	0	9	560.0	9	62.0
89	1	1	2	2	1	1	0	1	0	0	0	0	0	0	4	3	1	1	1	2	2	4	0	0	0	9	579.0	9	64.0
90	1	1	2	2	1	1	0	1	0	0	0	0	0	0	3	1	3	2	1	2	2	4	0	0	0	9	560.0	9	62.0
93	1	2	2	2	1	0	0	1	0	0	0	0	0	0	3	3	1	2	1	2	2	4	0	0	0	9	497.0	9	55.0
101	1	1	2	2	0	1	1	1	0	0	0	0	0	0	1	3	3	2	1	2	2	4	0	0	0	9	644.0	9	71.0
102	1	2	2	2	0	0	1	1	0	0	0	0	0	0	2	2	4	1	1	2	2	4	0	0	0	9	578.0	9	64.0

Figure 24: The subgroup with description *'register request' == 1 and 'check ticket' == 2 and 'reinitiate request' == 1*

Case_ID	register request	examine thoroughly	check ticket	decide	reject request	examine casually	pay compensation	reinitiate request	Pete	Sue	Mike	Sara	Sean	Ellen	Daniel	Martha	Erik	Chris	50	400	100	200	D0	D1	D2	D3	durHours	NoEvents	AEDIH
6	1	0	1	1	0	1	1	0	0	0	0	0	0	0	2	1	2	0	1	1	1	2	0	0	0	5	800.0	5	47.0
14	1	1	3	3	1	2	0	2	0	0	0	0	0	0	4	4	2	3	1	3	3	6	0	0	0	13	800.0	13	32.0
16	1	0	1	1	0	1	1	0	0	0	0	0	0	0	1	0	2	2	1	1	1	2	0	0	0	5	800.0	5	43.0
19	1	2	6	6	0	4	1	5	0	0	0	0	0	0	2	8	8	7	1	6	6	12	0	0	0	25	800.0	25	52.0
26	1	1	3	3	0	2	1	2	0	0	0	0	0	0	4	1	5	3	1	3	3	6	0	0	0	13	800.0	13	43.0
34	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	4	1	0	1	1	1	2	0	0	0	5	800.0	5	20.0
35	1	1	2	2	0	1	1	1	0	0	0	0	0	0	2	2	0	5	1	2	2	4	0	0	0	9	800.0	9	46.0
38	1	1	1	1	1	0	0	0	0	0	0	0	0	0	2	2	1	0	1	1	1	2	0	0	0	5	800.0	5	20.0
40	1	2	3	3	0	1	1	2	0	0	0	0	0	0	3	3	5	2	1	3	3	6	0	0	0	13	800.0	13	44.0
42	1	1	2	2	1	1	0	1	0	0	0	0	0	0	1	3	4	1	1	2	2	4	0	0	0	9	800.0	9	37.0
45	1	4	7	7	1	3	0	6	0	0	0	0	0	0	4	8	6	11	1	7	7	14	0	0	0	29	800.0	29	36.0
51	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	2	3	0	1	1	1	2	0	0	0	5	800.0	5	60.0
53	1	2	2	2	0	0	1	1	0	0	0	0	0	0	2	2	1	4	1	2	2	4	0	0	0	9	800.0	9	37.0
66	1	1	1	1	1	0	0	0	0	0	0	0	0	0	4	1	0	0	1	1	1	2	0	0	0	5	800.0	5	20.0
67	1	0	1	1	1	1	0	0	0	0	0	0	0	0	2	1	0	2	1	1	1	2	0	0	0	5	800.0	5	36.0
68	1	1	1	1	0	0	1	0	0	0	0	0	0	0	1	3	1	0	1	1	1	2	0	0	0	5	800.0	5	40.0
75	1	1	2	2	1	1	0	1	0	0	0	0	0	0	4	3	2	0	1	2	2	4	0	0	0	9	800.0	9	37.0
77	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	0	2	2	1	1	1	2	0	0	0	5	800.0	5	20.0
87	1	3	4	4	0	1	1	3	0	0	0	0	0	0	2	6	6	3	1	4	4	8	0	0	0	17	800.0	17	41.0
99	1	0	1	1	1	1	0	0	0	0	0	0	0	0	1	0	3	1	1	1	1	2	0	0	0	5	800.0	5	36.0
101	1	1	2	2	0	1	1	1	0	0	0	0	0	0	3	2	3	1	1	2	2	4	0	0	0	9	800.0	9	44.0

Figure 25: The subgroup with description *'register request' == 1 and 'Sara' == 0 and 'Ellen' == 0*

```
from pm4py.algo.filtering.pandas.attributes import attributes_filter

increasedE9D3 = attributes_filter.apply_events(increasedE9, ["D3"],
                parameters={attributes_filter.Parameters.CASE_ID_KEY:
                            "case:concept:name",
                attributes_filter.Parameters.ATTRIBUTE_KEY: "Department",
                attributes_filter.Parameters.POSITIVE: True})

alteredRSD3 = attributes_filter.apply_events(alteredRS, ["D3"],
              parameters={attributes_filter.Parameters.CASE_ID_KEY:
                          "case:concept:name",
              attributes_filter.Parameters.ATTRIBUTE_KEY: "Department",
              attributes_filter.Parameters.POSITIVE: True})
```

6.4.2 How should the results be transferred to a Process Mining tool?

It is now known that trace variable Department D3 is used to describe all the traces that we are interested in using our Exceptional Model Mining framework for both dataset. To be able to use this information, we need to filter the original event log such that it only contains traces that have this attribute. As mentioned in Section 6.3, we will use PM4PY for this, since our traces and the adapted dataset are already stored in Python format. The PM4PY library supports trace filtering on target attributes, so we use this to only select the traces that contain events that have been executed by someone in Department D3. We do so using the following function calls for both datasets respectively: Now we have event log *increasedE9D3*, which contains all traces of length nine, and an event log *alteredRSD3*, which contain traces with an execution time of 800 hours. These all contain events that were executed by an employee in department D3, which is exactly what we were looking for. We also generate event logs that contain all but the traces with events executed by Department D3, which can be seen as the complement of the adapted event logs. For this we used the following function calls:

Now we can use a process discovery algorithm on these adapted event logs and the original event logs to discover the underlying process model, and see whether they differ in e.g. performance.

6.4.3 Which information can we obtain from the generated process models?

For the six event logs we generated in Section 6.4.2, we can generate process models. We again use the PM4PY library to do this. When generating a process model, three variations can be created. The first one is just the plain process model, without any additional information. This process model is shown in Figure 2. It is also possible to add two types of information, namely, frequency and performance. The frequency information shows how often each event is

```
from pm4py.algo.filtering.pandas.attributes import attributes_filter

increasedE9D3Comp = attributes_filter.apply_events(increasedE9, ["D3"],
                parameters={attributes_filter.Parameters.CASE_ID_KEY:
                            "case:concept:name",
                attributes_filter.Parameters.ATTRIBUTE_KEY: "Department",
                attributes_filter.Parameters.POSITIVE: False})

alteredRSD3Comp = attributes_filter.apply_events(alteredRS, ["D3"],
               parameters={attributes_filter.Parameters.CASE_ID_KEY:
                           "case:concept:name",
               attributes_filter.Parameters.ATTRIBUTE_KEY: "Department",
               attributes_filter.Parameters.POSITIVE: False})
```

executed in the entire event log. The performance information shows how long each event takes to execute on average for the log. For all six obtained event logs, both the frequency and performance information containing process models have been generated using the Inductive Miner. Using the inductive miner with added frequency information on the the event log with increased traces of length nine, we obtain the process models found in Figure 26.

From these process models, some information can be obtained. We notice that the number of traces in the full event log in Figure 26a is 106, based on the fact that the *register request* event is performed 106 times. We also notice that approximately 40% of the cases have been reinitiated at least once for further investigation. and that the split between paying a compensation and rejecting the request is 50/50. It can also be seen that some of the requests have been reevaluated more than once, since the *decide* and *check ticket* events has been performed 183 times.
Next the event log that contains traces that have been performed by department D3 as seen in Figure 26b are inspected. We notice that there were 25 traces in this event log. All 25 of these have been reinitiated once, and for 56% of the cases a compensation has been paid.
Finally, we investigate the traces that were not performed by department D3, as seen in Figure 26c. The remaining traces from the original event log are shown here. We know that all cases that have been reinitiated in this event log, have been reinitiated at least twice, since the traces that were only reinitiated once are all in the traces performed by D3. This is the case for 18 of the traces. It is also noticed that the split between paying a compensation and denying a request is almost 50/50.

We can perform the same analysis on the process models generated by the inductive miner on the event log that contains traces that have an execution time of 800 hours. The corresponding models can be seen in Figure 27.

For all three event logs, the split between paying a compensation and rejecting the request is 50/50. In the original event log, for 43 traces the request has been reinitiated. For the event log containing traces performed by D3, this is the case for 11 traces. And in the event log of the remaining traces, this is the case for 32 of the traces. We also notice that the division between examining casually or thoroughly is almost equal for all three event logs.

These models have provided information about how often events have been executed in the traces in their corresponding event logs. It also shows information on how the traces are distributed between the different event logs. For the first event log, we knew that all traces of length nine were performed by D3, so we could make some assumptions on the remaining traces. However, this is not the most helpful information, since it does not really show us anything about the performance differences, which we know exist in the traces.

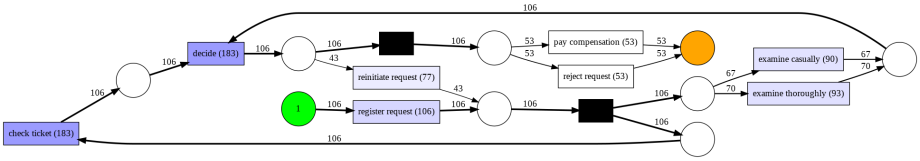
Next we analyse the generated process models with performance information included. The obtained process models for both event logs can be seen in Figures 29 and 28. When we look at the process model in 28a, we observe that it takes three days on average for the events *examine request, check ticket* and *decide*. It takes six days to pay the compensation or two days to reject the request after the decision has been made. When the request has to be reinitiated, this happens after one day on average.

When we look at the process model containing traces performed by department D3, we observe that for all but one events, the execution time has increased. Only the duration of *reinitiate event* is not visibly altered. This does not mean it is not increased, it just is not increased to take longer than one day, which can be up to 47 hours. We can conclude, that department D3 requires more time to complete their cases, but just to be sure, we also analyse the complement of the event log, as this contains all cases that were not executed by department D3. The corresponding model can be seen in Figure 28c. Here we observe that the average execution time per event indeed is lower for every event. We also observe that the reinitialize request is now decreased to 18 hours. We now reason that for the duration of *reinitiate request* for the entire event log to be at least 24 hours, it has to be at least 30 hours for the traces executed by D3, since $\forall_x \in 0 \leq x < 48 : \frac{18+x}{2} \geq 24$ has to hold.

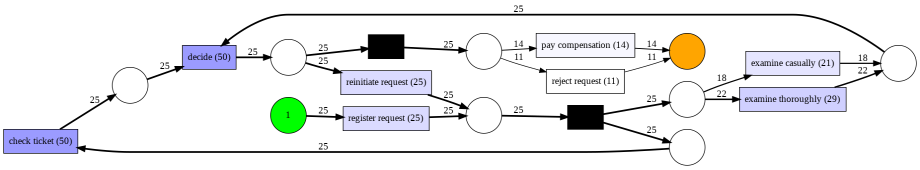
The process models, generated from the event log alterations containing traces that have a completion time of 800 hours for a random sample, are shown in Figure 29. When we compare the three models, we notice that the model that contains traces executed by D3 has a longer execution time for almost all events. Only *examine casually* is executed faster on average. This can be explained due to the fact that the execution time for the entire trace is set to 800 hours. For short traces, this meant an increase in duration per event. For long traces, this meant a decrease in duration per event. Long traces contain one or more loops, which means that *examine casually, examine thoroughly, check ticket* and *de-*

cide are performed more often per trace. They contribute more to the total execution time of the trace and are thus altered the most.
We see something similar happening for the *pay compensation* and *reject request* events of the traces, but the other way around. For every trace, these events can only be executed once. For the short traces, this results in a massive increase in execution time for these events, since these events cover 20% of the trace. For the longer traces, they cover only a small part, so their execution time is only decreased minimally. This results in a large increase in the average execution time for these two events, which we can see in the model.
In the model without traces executed by D3, we see mostly the same execution times for all events compared to the original model. The biggest difference is again observed in *reject request* and *pay compensation*, which makes perfect sense. In this model we do not include the traces executed by D3, which have a large execution time as we have seen before. This results in a lower average execution time for these two events.

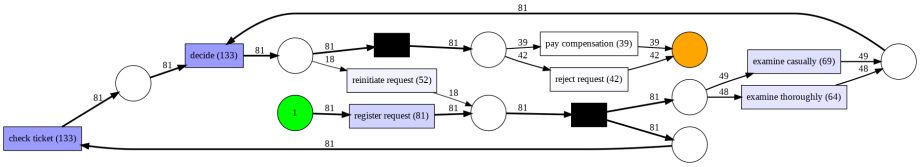
For both datasets, Exceptional Model Mining is able to discover subgroups that contain traces that show divergent behaviour. We can then analyse these subgroups and discover attributes that can be used to filter the original event log. From this event log we can generate process models using a miner found in the PM4PY library. Two variants of these process models contain frequency and performance information. When we analyse the information contained in the models, we see that there are differences between the models.
Our Exceptional Model Mining implementation is able to discover divergent and interesting behaviour from an altered event log and contain this behaviour in subgroups. The information in these subgroups can then be used to filter the original event log, which can then be further analysed using a Process Mining tool.



(a) Process model of the entire event log.

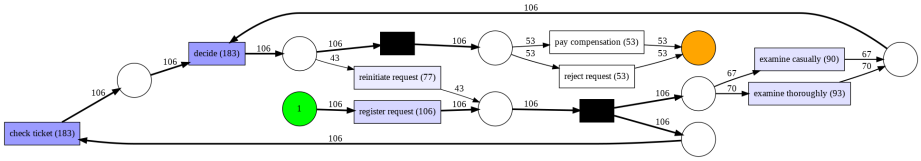


(b) Process model of the traces performed by D3.

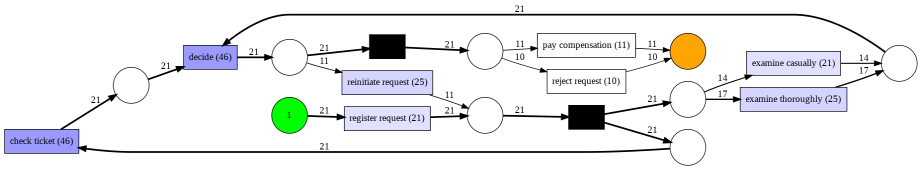


(c) Process model of the traces not performed by D3.

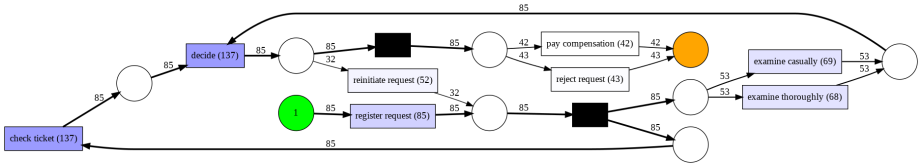
Figure 26: Frequency process models of the event logs with increased execution time for traces of length nine.



(a) Process model of the entire event log.

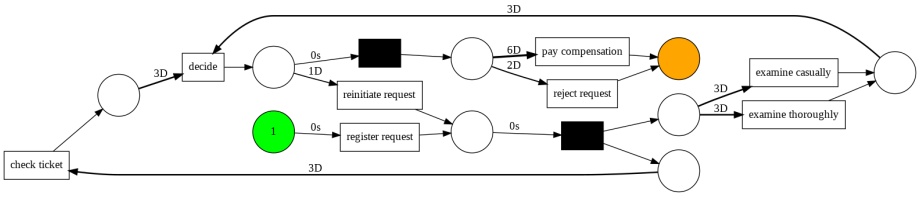


(b) Process model of the traces performed by D3.

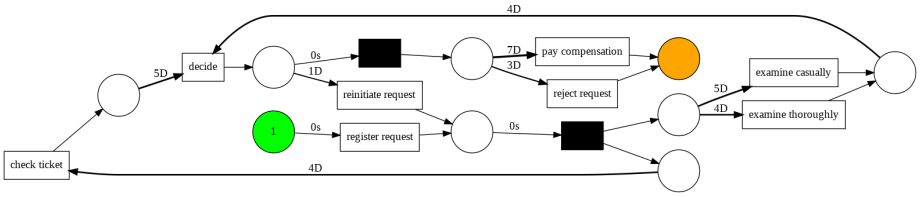


(c) Process model of the traces not performed by D3.

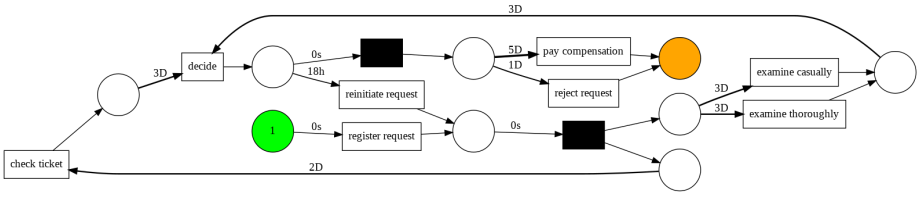
Figure 27: Frequency process models of the event logs with a duration of 800 hours for a sample of the traces.



(a) Process model of the entire event log.

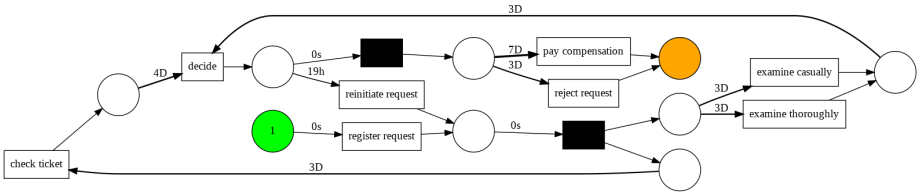


(b) Process model of the traces performed by D3.

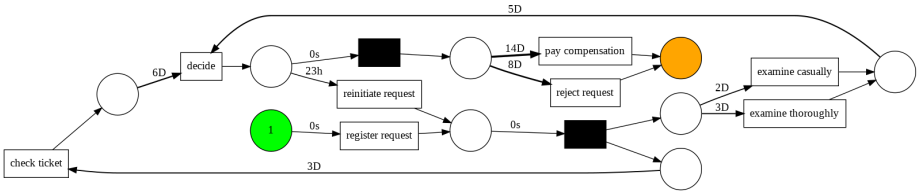


(c) Process model of the traces not performed by D3.

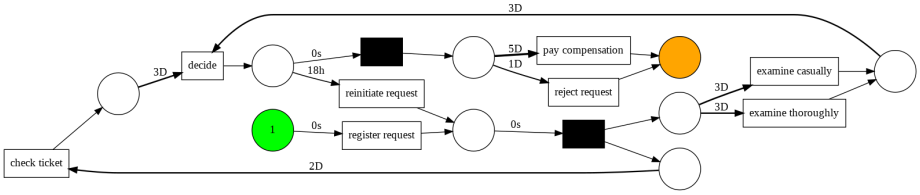
Figure 28: Performance process models of the event logs with increased execution time for traces of length nine.



(a) Process model of the entire event log.



(b) Process model of the traces performed by D3.



(c) Process model of the traces not performed by D3.

Figure 29: Performance process models of the event logs with a duration of 800 hours for a sample of the traces.

6.5 Discovery of results using traditional Process Mining techniques.

We have seen that we can use Exceptional Model Mining to discover subgroups containing cases with deviating behaviour from an event log. These subgroups can be obtained with very little prior process knowledge. However, all of this would be of little added value, if this is possible using already existing Process Mining techniques. First, we discuss and analyse the existing Process Mining techniques from Section 3.3. Secondly, we use the Dotted Chart view to analyse the cases in the event log. Since our adaptions are all performance based, we added *throughput time* and *event duration* attributes to the event log, using *Add Throughput Time as Trace Attribute (In place* and *Add Time between events (Duration) as Attribute to all Events* found in ProM. Since increasing performance is almost always desirable, prior process knowledge is not required to reason about the possible added value of this extra information.

6.5.1 Trace Clustering

Since trace clustering is capable of grouping traces together in clusters based on their similarities, we expected to be able to create clusters based on trace performance. When generating clusters based on *Department* and *Duration* dimensions, we were able to discover two clusters for both event logs. One cluster containing the cases executed by *D3*, the other cluster containing all other traces. For this event log, trace clustering is thus able to discover the same subgroup as Exceptional Model Mining. However, it should be noted that these clusters are found based on the department attribute. To verify this, we altered the event log again, such that the deviation in execution time for a subset of the cases is still present, but department *D3* is not assigned to these cases. Our Exceptional Model Mining implementation was able to discover this subgroup, as we can see in Figure 30. We can see in Figure 31 that this subgroup indeed does not have department *D3* involved, but that it still contains cases with length 9. As expected, we were not able to discover this subgroup using trace clustering.

6.5.2 Pattern Mining and Anomaly Detection

Both methods operate on process models, so we reason that they would not be possible to discover the correct subgroups. As we saw in Section 6.4.3, the process models for the subgroup and the entire event log are the same. This would mean that the cases with worse performance would not be discovered, since the event sequence remains the same for the deviating cases.

6.5.3 Dotted Chart

When exploring the event logs using the dotted chart visualization, we looked at the case distribution with *throughput time* on the y-axis and *Department* on the x-axis. The resulting plots can be seen in Figure 32. Here we can also clearly see the clusters of traces with deviating behaviour executed by department *D3*. This clustering of cases shows the same results as the clusters found using trace clustering. When analysing the dotted chart of the event log without *D3*, which is shown in Figure 32c, we also were unable to discover the deviating cluster, since *D3* does not exist anymore.

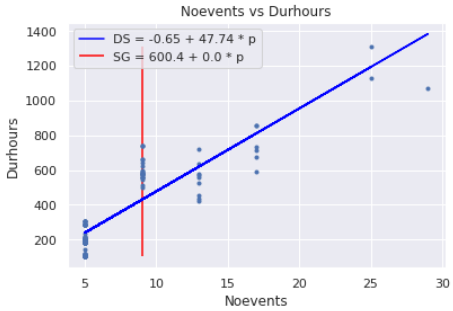
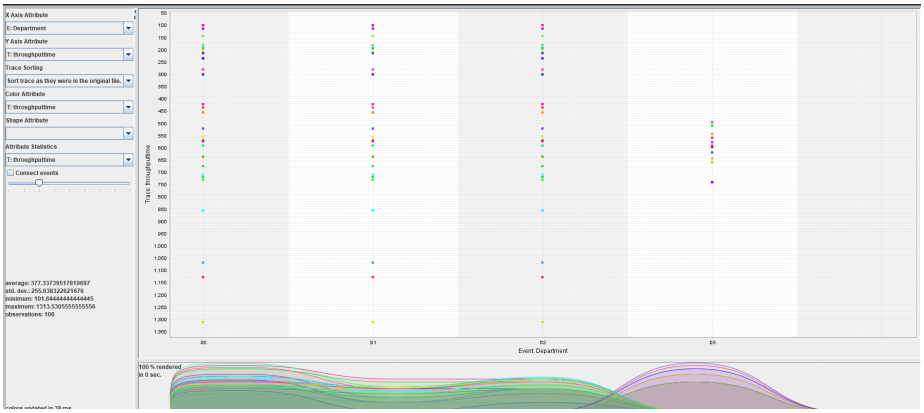


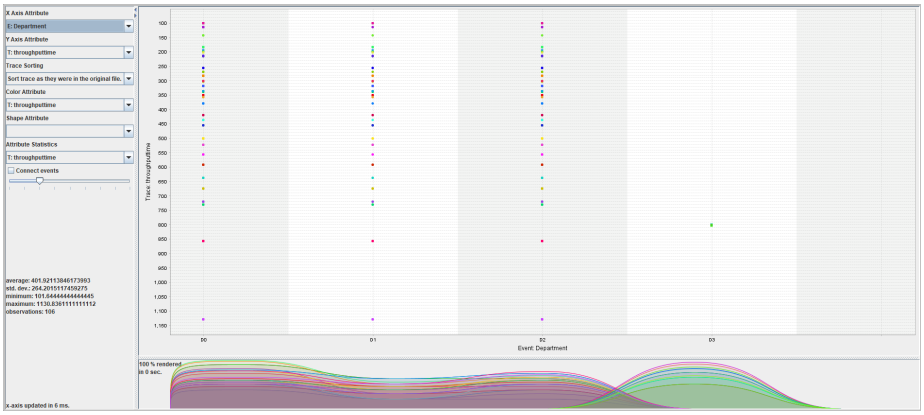
Figure 30: ‘check ticket‘ == 2 and ‘register request‘ == 1 and ‘reinitiate request‘ == 1

Case_ID	register request	examine thoroughly	check ticket	decide	reject request	examine casually	pay compensation	reinitiate request	Pete	Sue	Mike	Sara	Sean	Ellen	50	400	100	200	D0	D1	D2	durHours	NoEvents	AEDIH
3	1	1	2	2	0	1	1	1	2	0	1	3	1	2	1	2	2	4	5	1	3	620.0	9	68.0
7	1	1	2	2	1	1	0	1	1	0	0	3	2	3	1	2	2	4	4	2	3	560.0	9	62.0
8	1	2	2	2	0	0	1	1	2	2	0	3	0	2	1	2	2	4	4	2	3	578.0	9	64.0
24	1	1	2	2	1	1	0	1	0	0	2	3	1	3	1	2	2	4	5	1	3	511.0	9	56.0
33	1	2	2	2	0	0	1	1	1	1	2	3	1	1	1	2	2	4	4	2	3	578.0	9	64.0
35	1	1	2	2	0	1	1	1	0	1	2	3	0	3	1	2	2	4	5	1	3	660.0	9	73.0
39	1	1	2	2	1	1	0	1	1	0	2	3	2	1	1	2	2	4	4	2	3	560.0	9	62.0
41	1	0	2	2	0	2	1	1	1	0	4	3	1	0	1	2	2	4	5	1	3	742.0	9	82.0
42	1	1	2	2	1	1	0	1	2	1	1	3	1	1	1	2	2	4	4	2	3	579.0	9	64.0
46	1	1	2	2	0	1	1	1	1	1	1	3	1	2	1	2	2	4	4	2	3	592.0	9	65.0
53	1	2	2	2	0	0	1	1	0	2	0	3	0	4	1	2	2	4	4	2	3	578.0	9	64.0
55	1	0	2	2	0	2	1	1	0	1	3	3	0	2	1	2	2	4	5	1	3	742.0	9	82.0
61	1	1	2	2	0	1	1	1	0	1	0	3	1	4	1	2	2	4	4	2	3	660.0	9	73.0
62	1	2	2	2	0	0	1	1	1	0	2	3	2	1	1	2	2	4	4	2	3	578.0	9	64.0
70	1	0	2	2	0	2	1	1	1	1	4	3	0	0	1	2	2	4	5	1	3	742.0	9	82.0
71	1	1	2	2	0	1	1	1	1	1	2	3	1	1	1	2	2	4	4	2	3	592.0	9	65.0
75	1	1	2	2	1	1	0	1	0	2	2	3	0	2	1	2	2	4	4	2	3	579.0	9	64.0
76	1	2	2	2	1	0	0	1	3	1	0	3	1	1	1	2	2	4	4	2	3	597.0	9	66.0
78	1	1	2	2	1	1	0	1	2	0	1	3	1	2	1	2	2	4	5	1	3	544.0	9	60.0
83	1	1	2	2	1	1	0	1	1	1	1	3	1	2	1	2	2	4	4	2	3	560.0	9	62.0
89	1	1	2	2	1	1	0	1	2	1	1	3	1	1	1	2	2	4	4	2	3	579.0	9	64.0
90	1	1	2	2	1	1	0	1	2	2	2	3	0	0	1	2	2	4	4	2	3	560.0	9	62.0
93	1	2	2	2	1	0	0	1	2	0	1	3	2	1	1	2	2	4	4	2	3	497.0	9	55.0
101	1	1	2	2	0	1	1	1	0	0	4	3	1	1	1	2	2	4	5	1	3	644.0	9	71.0
102	1	2	2	2	0	0	1	1	2	0	1	3	2	1	1	2	2	4	4	2	3	578.0	9	64.0

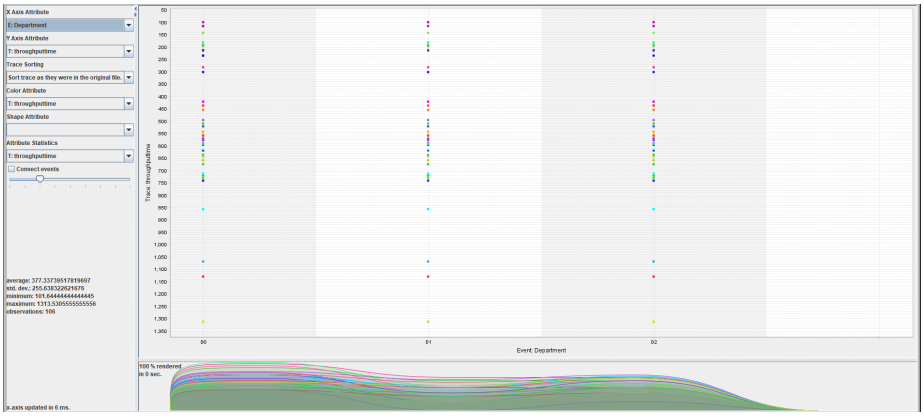
Figure 31: The subgroup with description ‘check ticket‘ == 2 and ‘register request‘ == 1 and ‘reinitiate request‘ == 1



(a) Event log with increased duration for traces of length 9.



(b) Event log with altered duration for sample of the traces.



(c) Event log without department *D3*.

Figure 32: Dotted Chart view for the three event logs, with throughput time and department information on the axes.

7 Conclusion

During this thesis, we have explored and investigated the possibilities of using an Exceptional Model Mining framework implementation to aid us in the Process Mining process discovery process to be able to answer our research question, *Can Exceptional Model Mining be used in Process Mining to discover subsets of traces that contain deviating behaviour based on user specified attributes?* To be able to answer this question, subquestions have been formulated and answered to guide us through the entire process.

7.1 What did we achieve?

How can we obtain traces that contain deviating behaviour from event logs using Exceptional Model Mining?
As seen in Section 4.1, when Exceptional Model Mining is used to discover subgroups containing divergent behaviour, the corresponding descriptions should be used to prune the dataset. This should be done because the descriptions describe subgroups containing divergent behaviour, but might not be very helpful at first, because they seemingly do not make a lot of sense. When pruning the dataset, the other attributes are also shown. These attributes might show a more understandable characteristic of the subgroup. E.g., the description *'register request' == 1 and 'check ticket' == 2 and 'reinitiate request'== 1* (The description of Figure 16a) does not directly describe why the items in the subgroup are divergent. But when we use this description to prune the dataset, we notice that department *D3* is involved in the execution of all these cases. The obtained results can then be used to prune the original event log, to obtain the traces that contain this divergent behaviour. These can be further analysed using Process Mining techniques. E.g., a process model can be generated from the pruned event log, which can be shown to and discussed with the process owner.

How should we transform the event logs?
In Section 5.1, we saw that Process Mining operates on event logs. These event logs contain traces, which contain trace attributes, the sequence of events, and the corresponding event attributes. When stored in a matrix format, these are stored as an event per row, with a case identifier and their other events in the corresponding columns. Exceptional Model Mining operates on datasets. These datasets are also stored in matrix format, where each row is an item which is described by its attributes stored in the corresponding columns. We are interested in traces as a whole and observe that there is a mismatch between the two data formats. Exceptional Model Mining operates on items stored in single rows, containing all the information of that item in its columns. An event log spreads the information of a trace over multiple rows, where each row contains an event and the corresponding event attributes. This way the sequence of events is clearly visible. The need to transform the event log arises, where every row describes a single trace, the activities of that trace, and the other event and trace attributes.

Which information should these transformed traces contain?
As seen in Section 5.2, the transformed case-attribute table should contain as much information as possible of the original event log. In every row, describing one trace, the case identifier, the trace attributes, the events and the event attributes should be stored. When storing trace attributes and the case identifier, no information is lost, since this is the same for the entire trace. When storing the events and the event attributes, we do lose some information, since we only count the occurrences per trace. The sequence of events in a trace and the information on which attributes belong to which event is lost. This is desired knowledge to have in Process Mining, as the discovery of a correct process model relies heavily on this sequence. Another feature of Process Mining is the discovery of loops. Since we lost the event sequence in our new data format, this information cannot be directly obtained. However, one can argue that when an event occurs more than once in a trace, this is a good indication of the presence of a loop. For our proposed implementation, this loss of information turned out to be no issue. We were able to discover subgroups showing deviating behaviour and find an attribute that can be used to filter the original event log. The traces in this filtered event log contained all information lost during the initial transformation, which made it possible to further analyse the event log using Process Mining.

How should the results obtained by Exceptional Model Mining be analysed to obtain useful insights? In Section 6.1 the quality measures that were used and the alterations to the original event log were explained. We decided to use *linear regression*, *Euclidean distance* and *z-score* as quality measures. The event log was altered by increasing the execution time for some of the traces, and assigning them to a new department *D3*, which makes it possible to discover these using Exceptional Model Mining. Section 6.2 contains the obtained results for the event log that contains traces of length 9 that have an increased execution time. Section 6.3 contains the obtained results for the event log that contains uniformly at random selected traces that have an execution time of 800 hours. Since we only used two target attributes, we were able to plot the regression lines for both the traces in the subgroup and the traces in either the complement of, or the complete event log. This visualization made the interpretation of the results easier. As seen in Figures 16 and 22, the *z-score* quality measure performs best and is able to discover the deviating traces for both event logs.
The descriptions that were able to describe the deviating subgroups correctly, were used to prune the dataset to obtain the corresponding subgroup. These subgroups are analysed for attributes that could be used to filter on the original event logs.

How can these interpreted results help us with Process Mining?
As seen in Section 6.4, we used the top description from the *z-score* quality measure to prune the case-attribute table for both the adaptions we made. The corresponding subgroups were analysed for attributes that could be used to fil-

ter on the original event logs. We filtered these event logs, such that for each alteration, we ended up with three event logs. One that contains all traces with altered execution times, one that contains all but the traces with altered execution time (the complement), and finally the complete event log. For these three event logs, we generated the process models using the Inductive Miner. In these process models, we included the frequency of the events, as seen in Figures 26 and 27 or the performance of the events, as seen in Figures 28 and 29. After analysing these process models, we were able to conclude that the performance of the traces executed by department *D3* differs from the performance of the rest of the traces. This discovery is the same as the discovery we made using our Exceptional Model Mining framework.

7.2 Are we satisfied with the results?

The results we obtained are very promising, it is a good indication that Exceptional Model Mining can be used for Process Mining. We are able to discover subgroups that show the interesting behaviour that was manually added to the original event log. The subgroups are analysed and some of the attributes are used to filter the original event log. From this event log, a process model could be generated which shows the same deviating behaviour as the corresponding subgroup.

It should be noted, however, that the original event log describes a very simple process and is correct for every trace. This means that a process model generated from this data always correctly describes the underlying process. This is almost never the case in real life situations. It also makes the analysis using Process Mining tools very straightforward.

One should also take into account that the alterations that were made, were relatively simple, since they are only based on two deviating target attributes. However, this is not immediately an issue, since the user might only be interested in two attributes. It is possible to select more attributes as target attributes, which does not alter the rest of the subgroup discovery process. The only thing that does change, is that the *regression* quality measure can no longer be used, since this can only operate on two target attributes.

That the used data and alterations are relatively simple, is not immediately an issue. The foundation of the application of Exceptional Model Mining in Process Mining is laid, and the topic is open for further research. The initial results are promising, and we are positive that Exceptional Model Mining will be able to provide some form of assistance when analyzing a process using Process Mining.

7.3 What can still be done?

In the previous section, we already implied what can and should be done in the future. In this thesis, we used an event log, which describes a simple process and is always correct. This almost never happens in real life processes. Therefore, for future work, it would be interesting to investigate how well our Exceptional Model Mining framework performs on real life data, and how much has to be added and altered to the current implementation to make it work.

The ability to store the relations between the events when transforming the event log would also be convenient. This information is now lost in the conversion process and cannot be retrieved once the conversation is done. However, the original event log is stored, so we can apply filters to this log to obtain the correct traces. This worked for our research, but it can happen that the descriptions and the corresponding subgroup do not contain attributes that can be used for filtering the event log, which is why it would be nice to be able to transform the Case Attributes table back to the original event log. This would require a lot of extra, for now unnecessary storage, which is why we did not include this. A quick and easy workaround is filtering the original event log using the case identifier, since this is unique for every trace, and is stored in the Case Attribute table.

It would also be interesting to see how our Exceptional Model Mining framework would perform when more complex target models are chosen, e.g., when selecting three or more target attributes. The addition of more quality measures to evaluate the descriptions would also be good practice. The selection of the target model and which quality measure is being used, is completely up to the user. So if the user desires a different quality measure, he is free to do so.

Another desirable feature for our Exceptional Model Mining framework would be the use of $<, >, \leq, \geq$ operators, when generating the descriptions. We started only with the equality operator and managed to get everything to work with this. However, we were running out of time, so we were unable to include the other operators and test everything again. This should however, only be a few hours work to get it functional, after which the testing can begin.

References

[1] Wil van der Aalst. "Data Science in Action". In: *Process Mining: Data Science in Action*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 55–88. ISBN: 978-3-662-49851-4. DOI: 10.1007/978-3-662-49851-4_1. URL: https://doi.org/10.1007/978-3-662-49851-4_1.

[2] Wil van der Aalst et al. "Process Mining". In: (2016). URL: http://www.processmining.org/event_logs_and_models_used_in_book.

[3] Abel Armas-Cervantes et al. "Local Concurrency Detection in Business Process Event Logs". In: *ACM Trans. Internet Technol.* 19.1 (Jan. 2019). ISSN: 1533-5399. DOI: 10.1145/3289181. URL: https://doi.org/10.1145/3289181.

[4] Adriano Augusto et al. "Split miner: automated discovery of accurate and simple business process models from event logs". In: *Knowledge and Information Systems* 59 (May 2019). DOI: 10.1007/s10115-018-1214-x.

[5] Alessandro Berti, Sebastiaan J. van Zelst, and Wil M. P. van der Aalst. "Process Mining for Python (PM4Py): Bridging the Gap Between Process- and Data Science". In: vol. abs/1905.06169. 2019. arXiv: 1905.06169. URL: http://arxiv.org/abs/1905.06169.

[6] Wouter Duivesteijn. "Exceptional Model Mining". In: *Doctoral Thesis, Leiden University* (Sept. 2013), pp. 1–26, 33–40.

[7] Wouter Duivesteijn. "Exceptional Model Mining". In: *Doctoral Thesis, Leiden University* (Sept. 2013), p. 15.

[8] Wouter Duivesteijn, Ad J. Feelders, and Arno Knobbe. "Exceptional Model Mining". In: *Data Min. Knowl. Discov.* 30.1 (Jan. 2016), pp. 47–98. ISSN: 1384-5810. DOI: 10.1007/s10618-015-0403-4. URL: https://doi.org/10.1007/s10618-015-0403-4.

[9] M. L. V. Eck et al. "PM^2 : A Process Mining Project Methodology". In: *CAiSE*. 2015.

[10] Dirk Fahland and Wil M.P. van der Aalst. "Model repair — aligning process models to reality". In: *Information Systems* 47 (2015), pp. 220–243. ISSN: 0306-4379. DOI: https://doi.org/10.1016/j.is.2013.12.007. URL: https://www.sciencedirect.com/science/article/pii/S0306437913001725.

[11] Mohammadreza Fani Sani et al. "Subgroup Discovery in Process Mining". In: May 2017, pp. 237–252. ISBN: 978-3-319-59335-7. DOI: 10.1007/978-3-319-59336-4_17.

[12] H.S. Garcia Caballero et al. "Visual analytics for soundness verification of process models". English. In: 6th International Workshop on Theory and Application of Visualizations and Human-centric Aspects in Processes, TaProViz'17 ; Conference date: 11-09-2017 Through 11-09-2017. 2017. URL: http://www.wst.univie.ac.at/topics/taproviz17/.

[13] Laura Genga et al. “Discovering anomalous frequent patterns from partially ordered event logs”. In: *Journal of Intelligent Information Systems* 51.2 (2018), pp. 257–300.

[14] Francisco Herrera et al. “An overview on subgroup discovery: Foundations and applications”. In: *Knowledge and Information Systems* 29 (Dec. 2011), pp. 495–525. DOI: 10.1007/s10115-010-0356-2.

[15] Petra Kralj Novak, Nada Lavrac, and Geoffrey Webb. “Supervised Descriptive Rule Discovery: A Unifying Survey of Contrast Set, Emerging Pattern and Subgroup Mining.” In: *Journal of Machine Learning Research* 10 (Jan. 2009), pp. 377–403. DOI: 10.1145/1577069.1577083.

[16] Petra Kralj Novak et al. “CSM-SD: Methodology for contrast set mining through subgroup discovery”. In: *Journal of Biomedical Informatics* 42.1 (2009), pp. 113–122. ISSN: 1532-0464. DOI: https://doi.org/10.1016/j.jbi.2008.08.007. URL: https://www.sciencedirect.com/science/article/pii/S1532046408001032.

[17] Nada Lavrač, Peter Flach, and Blaz Zupan. “Rule Evaluation Measures: A Unifying View”. In: *Inductive Logic Programming*. Ed. by Sašo Džeroski and Peter Flach. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 174–185. ISBN: 978-3-540-48751-7.

[18] Sander J. J. Leemans, Dirk Fahland, and Wil 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*. Ed. by José-Manuel Colom and Jörg Desel. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 311–329. ISBN: 978-3-642-38697-8.

[19] Sander JJ Leemans. “Robust process mining with guarantees”. PhD thesis. 2018, pp. 46–50.

[20] Jelle Marius. “Process Mining, laten we bij het begin beginnen”. In: (Mar. 2018). URL: https://bureautromp.nl/process-mining-laten-we-begin-beginnen/.

[21] Barbara FI Pieters, Arno Knobbe, and Sašo Dzeroski. “Subgroup discovery in ranked data, with an application to gene set enrichment”. In: (2010).

[22] S. Shankar and T. Purusothaman. “Utility Sentient Frequent Itemset Mining and Association Rule Mining: A Literature Survey and Comparative Study”. In: *International Journal of Soft Computing Applications* (Jan. 2009).

[23] Minseok Song, Christian Günther, and Wil Aalst. “Trace Clustering in Process Mining”. In: *Lecture Notes in Business Information Processing* 17 (Sept. 2008), pp. 109–120. DOI: 10.1007/978-3-642-00328-8_11.

[24] Niek Tax, Natalia Sidorova, and Wil Aalst. “Local Process Models: Pattern Mining with Process Models”. In: (June 2017).

[25] Irene Teinemaa et al. "Outcome-Oriented Predictive Process Monitoring: Review and Benchmark". In: *ACM Trans. Knowl. Discov. Data* 13.2 (Mar. 2019). ISSN: 1556-4681. DOI: 10.1145/3301300. URL: https://doi.org/10.1145/3301300.

[26] Willy Ugarte et al. "Skypattern mining: From pattern condensed representations to dynamic constraint satisfaction problems". In: *Artificial Intelligence* 244 (2017). Combining Constraint Solving with Mining and Learning, pp. 48–69. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2015.04.003. URL: https://www.sciencedirect.com/science/article/pii/S000437021500065X.

[27] H.M.W. Verbeek et al. "ProM 6 : the process mining toolkit". English. In: CEUR Workshop Proceedings (2010). Ed. by M. La Rosa, pp. 34–39.

[28] A.J.M.M. Weijters, W.M.P. Aalst, van der, and A.K. Alves De Medeiros. "Process mining with the HeuristicsMiner algorithm". English. In: BETA publicatie : working papers (2006).

[29] Wikipedia contributors. *Data mining — Wikipedia, The Free Encyclopedia*. [Online; accessed 02-December-2020]. 2020. URL: https://en.wikipedia.org/wiki/Data_mining.

[30] Wikipedia contributors. *Euclidean Vector — Wikipedia, The Free Encyclopedia*. [Online; accessed 09-December-2020]. 2020. URL: https://en.wikipedia.org/wiki/Euclidean_vector.

[31] Wikipedia contributors. *Simple linear regression*. [Online; accessed 10-February-2021]. 20021. URL: https://en.wikipedia.org/wiki/Simple_linear_regression.