universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Process Enhancement inspired by Key Performance Indicators

verfasst von | submitted by

Simon Strasser BSc (WU)

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2024

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 926
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Wirtschaftsinformatik
Betreut von | Supervisor:	Mag. Dipl.-Ing. Dr.techn. Maria Leitner Bakk.

Abstract

Process mining is a powerful tool for process owners and analysts by offering valuable insights and aiding in improvements. One part of improvement is process enhancement which in turn uses model mepair to adapt an existing process model towards better performance. The goal of this thesis is to utilize Key Performance Indicators (KPI) to improve the performance of process behavior which is represented in models. The focus is on manufacturing data which is a rarely discussed field in the context of process mining. Two extensions to existing model repair techniques are proposed to provide a tailored approach to model repair on manufacturing processes. One extension is named *avoid-flower* and focuses on precise repaired models. The other - *move-loc* - addresses feature selection when learning the pattern between KPI value and trace behavior. The extensions are implemented in a Python application. Case study data from manufacturing organizations and administrative processes is analyzed aiming to improve an original model by adding desirable process behavior. The results show that *move-loc* is suitable for adapting an original model better to test data and achieving better KPI values than model repair without this extension. On the other hand, *avoid-flower* creates simple repaired models and is suitable for processes with low variability. Furthermore, this thesis discusses additional versions of model repair, the role of KPIs and other process model quality dimensions.

Abstract in German

Process Mining ist ein leistungsfähiges Instrument, das Prozessverantwortlichen und -analysten wertvolle Erkenntnisse liefert und Verbesserungen unterstützt. Ein Teil der Verbesserung ist Process Enhancement, die wiederum Model Repair nutzt, um ein bestehendes Prozessmodell hinsichtlich eines bestimmten Ziels anzupassen. Die in dieser Arbeit verwendete Methodik berücksichtigt Key Performance Indicators (KPI), um die Leistung des Prozessverhaltens, das in einem Modell dargestellt wird, zu verbessern. Der Schwerpunkt liegt dabei auf Fertigungsdaten, einem im Kontext des Process Mining selten behandelter Bereich. Es werden zwei Erweiterungen zu bestehenden Model-Repair-Techniken vorgestellt, um einen maßgeschneiderten Ansatz für die Reparatur von Fertigungsprozessen zu bieten. Eine Erweiterung hat den Namen *avoid-flower* und zielt auf präzise reparierte Modelle ab. Die andere wird *move-loc* genannt und befasst sich mit der Auswahl der Merkmale, die beim Erlernen des Musters zwischen KPI-Wert und Trace-Verhalten herangezogen werden. Die Erweiterungen sind in einer Python-Anwendung implementiert. Fallstudiendaten aus Fertigungs- und Verwaltungsprozessen werden analysiert, um ein bestehendes Modell durch Hinzufügen von erwünschtem Prozessverhalten zu verbessern. Die Ergebnisse zeigen, dass *move-loc* geeignet ist, ein Originalmodell besser an die Testdaten anzupassen und bessere KPI-Werte zu erzielen als Model Repair ohne diese Erweiterung. Demgegenüber erzeugt *avoid-flower* einfache reparierte Modelle und ist für Prozesse mit geringer Variabilität geeignet. Darüber hinaus werden in dieser Arbeit weitere Varianten von Model Repair, die Rolle von KPIs und andere Aspekte der Modellqualität behandelt.

Acknowledgements

Diese Masterarbeit ist im Zusammenwirken mit einem von der FFG geförderten Forschungs- und Entwicklungsprojekt entstanden (www.ffg.at).

Contents

1 Introduction

One main goal of contemporary businesses is continuous process improvement. However, in the plethora of data tracked and stored in organizations today, it is increasingly difficult to find the needle in the haystack that is the key to process optimization. To unlock the true potential of efficient Business Process Management (BPM), organizations must embark on a journey of profound process understanding. This journey often starts with the application of Six Sigma methodology, a robust framework designed to eliminate process defects and enhance operational efficiency [2]. Comprising five distinct phases — define, measure, analyze, improve, and control — Six Sigma equips organizations with a systematic approach to identifying, measuring, and mitigating process anomalies. However, frameworks can only build the basis for analyses that rely on data captured in the daily business routine, such as process log data. Concrete performance measure is another ingredient to purposeful BPM.

Key Performance Indicators (KPIs) play an instrumental role in modern organizations, serving as guidance on the path to both current success and future prosperity [3]. KPIs are described as "a set of measures focusing on those aspects of organizational performance that are the most critical for the current and future success of the organization". These metrics are not mere data points; they represent operations within an organization to a very detailed level, providing insights that are critical for steering businesses toward their goals. Within manufacturing industry, the incorporation of KPIs assumes a role of high significance. These metrics empower organizations not only to monitor and optimize their internal operations over time but also to engage in meaningful benchmarking against similar enterprises within their industry [4]. Moreover, KPIs fuel effective decision-making, providing leaders with the data-driven insights needed to navigate the complex landscape of modern business [5]. The use of KPIs for business process management requires the understanding of which processes in an organization exist and how their performance can be measured.

In the ever-evolving field of business analytics, process mining emerges as a prominent and dynamic discipline, offering a multitude of techniques and concepts for the comprehensive analysis of business data [6]. It is the art of data-driven discovery, utilizing event data to construct process models that shine a spotlight on how a process unfolds [7]. Process mining takes event data as input, creates process models and aims to provide valuable insights about how a process is executed [7]. Its main objective is to provide actionable insights into process execution, enabling organizations to identify bottlenecks, deviations from planned process models, and even predict performance and compliance issues.

One of the initial mental images that might come to one's mind when thinking of process mining is that of a graphic representation portraying a process model. This visual depiction serves as a powerful starting point for discussions and analyses, providing a tangible and intuitive framework for domain experts and process mining engineers to dissect and explore. This graphic representation encapsulates the essence of a particular process, offering a dynamic and interactive way of looking into an organization's operations. The process model serves as a common language, bridging the communication gap between stakeholders from various backgrounds and expertise levels. Whether it is a domain expert from the industry or a process mining engineer, this visual representation acts as a universal canvas upon which ideas, insights, and strategies are sketched, refined, and perfected. It transforms complex, data-rich processes into comprehensible images that facilitate a shared understanding among all parties.

In the vast landscape of process modeling, there exist several commonly adopted modeling languages, each tailored to specific needs. Business Process Modelling Notation (BPMN) and Petri nets are two commonly used process modeling languages. They serve as the basis of process

modeling throughout a broad spectrum of applications and industries. BPMN is a visual language with a focus on representing business processes with clarity and precision [8]. Its standardized symbols and notation conventions provide a common framework for modeling processes, making it an ideal choice for organizations seeking a structured and universally understood approach to process modeling.

On the other hand, Petri nets are a mathematical modeling language and offer a more formal and rigorous approach to process representation [9]. Petri nets are particularly well-suited for scenarios where precise modeling of concurrency, synchronization, and causality is paramount. They excel in capturing complex interactions and dependencies within processes, making them highly valuable in industries where safety, reliability, and performance are critical factors. The choice between BPMN and Petri nets often hinges on the specific needs and objectives of a process mining project. BPMN's intuitive visual language is in favor when the emphasis is on business process transparency and ease of communication. Conversely, Petri nets are eminently suitable when the analytical depth and mathematical rigor of the model are important. Due to the focus on analytic of deviations between historic data and existing models, Petri nets are used for the scope of this thesis in order to fulfill the requirements.

Within the expansive realm of process mining, *process enhancement* emerges as a specialized domain, focused on the augmentation and refinement of existing models through insights extracted from recorded event log data [6]. While process enhancement goes beyond mere analysis, its ultimate goal is to recommend adaptive changes to processes that lead to improved KPI values [1]. However, it presents a unique challenge — the identification of KPIs that are not only desirable from a domain perspective but can also be seamlessly integrated into the process enhancement workflow. The relationship between process execution and resulting KPI values is one of the challenges tackled in this thesis. In particular, it is necessary to align a KPI as target variable to different executions of a process and find out how the way of execution and the KPI value might be connected.

Based on the existing example of KPI-based model repair presented by Dees et al. [1], this thesis proposes two extensions to KPI-based model repair. One of them - *avoid-flower* - is designed to obtain clearer, more precise repaired models and the other one - *move-loc* - emphasizes a more thorough decision making in the learning problem that separates desirable from undesirable process instances. These extensions enhance the existing repair technique and the results are compared to the results of the "original" technique by Dees et al. [1]. Moreover, we compare repaired models to models that are discovered based on repaired log. For this purpose, we apply widely-used discovery algorithms, namely inductive miner and heuristics miner. While model repair aims to stay as close as possible to the original model while adding behavior, discovery creates a whole new model and balances model complexity and fitness to the log.

KPI-based process enhancement is a transformative approach to process modeling. It impacts existing process models, with a focus on key model quality dimensions: fitness, precision, simplicity, and generalization. Furthermore, the similarity between the original model and the repaired model is considered by simply counting the conducted repair steps. By focusing on these key model quality dimensions, organizations can harness the power of KPIs to elevate their existing process models to new heights. This approach not only enhances operational efficiency but also ensures that process models remain agile and adaptable in the face of evolving business challenges.

One focus of this thesis is model repair in the field of manufacturing processes. In current literature, most case studies deal with administrative processes. However, it is sensible to analyze manufacturing processes in this context. Usually there are planned production processes where one step follows the next following a work schedule that is planned beforehand. Businesses in manufacturing industry may have an interest to not only check for conformance between model

and log but also to gain insights based on automated suggestions for process adaption. KPI-based Model repair can meet this need by adding behavior of desirable process instances to the existing model.

Business organizations are not only interested in the use of KPIs to control their performance and hence their success, but also to utilize process mining and the wide range of analytical methods and insights in business activities that come with it. Organizations aspire to leverage the vast potential of process mining and its accompanying analytical methods. Bridging the gap between KPIs that are widely used in manufacturing and those that are common to measure business process performance is one of the goals of this thesis. The progress made in aligning process models with KPIs is showcased, thereby enhancing real-world processes within organizations in a meaningful and efficient manner. KPI-based process enhancement marks another data science application that enables manufacturing businesses to gain a clearer picture and to extract actionable insights about process improvement in the deluge of data that is tracked and captured in organizations today.

This thesis is structured as follows: Chapter 2 describes the basics of process mining regarding procedures and algorithms as well as graphical and logical representation. The general idea of model repair is presented in chapter 3 using the notion of alignment steps and based on KPI values. Chapter 3 also includes the related work. The subsequent chapter 4 describes how process mining is utilized in the scope of this thesis and introduces the implementation. The introduction and evaluation of case study data is described in chapter 5, which is followed by the conclusion and outlook in chapter 6.

2 Preliminaries

Before describing how we meet the various challenges posed in the introduction, we first introduce some process modelling and process mining notation and operators. Moreover, model quality dimensions are discussed.

2.1 Process Mining Basics

To start with, the core concepts of process mining are briefly described. A process can be described as a finite set of subsequent activities (i.e. steps in the process) that are carried out to complete a task or achieve some other sort of goal. In the context of process mining, the concept *event log* is used to summarize the important aspects [10]. For process mining, the execution of a process is tracked in form of a sequence of *events*. An event is the representation of an activity in the event log. Each event belongs to a particular *trace* (also referred to as *case*) which is the representation of one process instance. So a log consists of traces which themselves consist of events. Events must have an attribute that represents their sequence allows to sort them. Typically this is a time stamp but it could be a consecutive number as well. An event can carry additional attributes such as resource (person, machine, etc.) that enable further analyses.

Process mining can be divided into three types, namely process discovery, conformance checking and process enhancement. **Process discovery** is the term for techniques that create the process model from recorded process log data [7]. There are various algorithms for process discovery. How process discovery works is shown in a simple example Petri net model in Figure 2.1. It is created from the event log displayed on the left-hand side and is based on an example by van der Aalst [11]. The two lines inside the cylinder on the left side represent the traces. Generally, process discovery aims to build a process model that represents all traces from the log data the model is built upon. Differences between traces therefore play a role which typically leads to a model that does not include each and every trace but only the most frequent ones if the input is a lot of log data, with much variation between traces. In the context of process mining, *noise* describes infrequent and missing or faulty logs of process instances that can be omitted for process discovery.

In the example depicted in 2.1, a model is discovered based on only two traces which makes it trivial to include both of them. The process has been captured as a sequence of *a, b* and *c* but also as a sequence of *a, b* and *d*. The model includes an exclusive choice after transition *b* which means that either *c* or *d* is executed but not both of them.

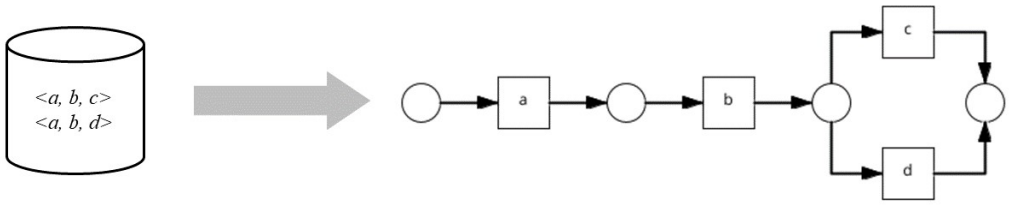


Figure 2.1: Process discovery: example petri net from event log

Once a process model has been created, **conformance checking** can be applied to compare the model to the process data from recorded during process executions in the “real world" [12]. The model represents how the process is intended to be executed and the recorded process data

represents how it is actually executed. The motivation is to validate if the process model describes reality and if the recorded log data is compliant. It can also support prediction-making about running cases (e.g. remaining time/cost).

The third type, **process enhancement**, refers to process models that need to be precise and realistic and also fulfill laws and regulations (i.e. ensure compliance) [13]. Process enhancement can be further divided into *process extension* and *process improvement*. Process extension adds perspectives on data, decision, resources and time into the model and allows for fine-tuning and hence more precise models. Process improvement, on the other hand, removes or adds portions of behavior in order to avoid unsatisfactory outcomes (high costs, low customer satisfaction, violation of regulations, etc.) Since process mining is described as the intersection of data science and process science [6], it is of interest for organizations to utilize data science concepts and tools in the context of process improvement to obtain more desirable processes and models. Figure 2.2 illustrates process discovery, conformance checking and process enhancement and relates it to the "real world" and the software system inside an organization.

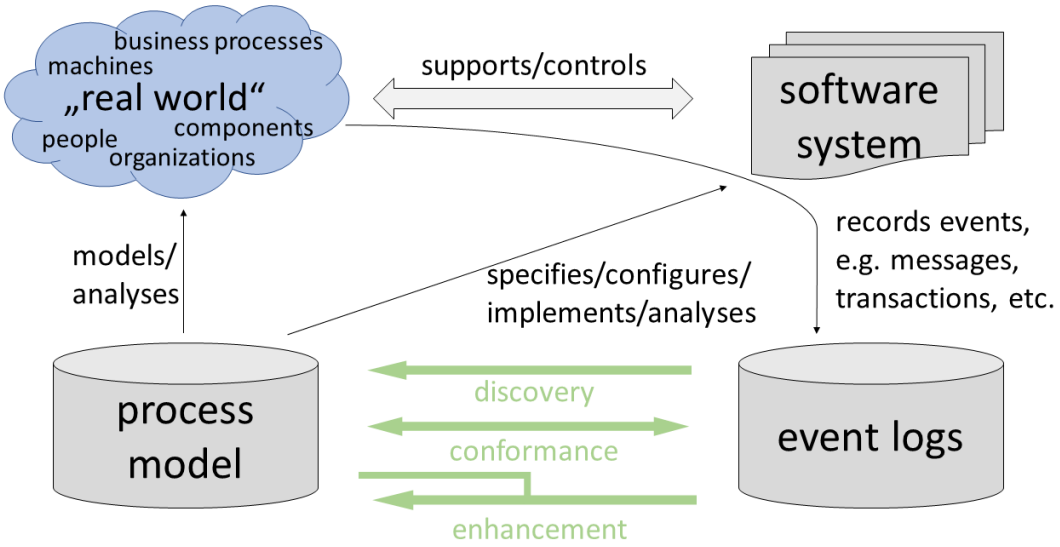


Figure 2.2: Position of the three process mining types: discovery, conformance and enhancement [6]

Although the focus of this thesis is on model repair which has fundamental differences to process discovery, a couple of process mining algorithms is discussed. This is due to the following reasons: Firstly, we want to point out the differences in the driving forces of process discovery and model repair and will compare the result of process discovery algorithms applied on a repaired log to the result of model repair using the repaired log. Secondly, for the very procedure of model repair, process discovery is required at the point when a sub-model needs to be created that is integrated into an existing model in a subsequent step. Thirdly, process discovery plays a role in the identification of the original process model that is the basis for alignment analysis. Here the goal is to create a model to start with that includes all "standard" behavior. At the same time, the starting model must leave room for deviations in the log that happen frequent so that it makes sense to repair them as they are no outliers or noise.

2.1.1 Process Discovery Algorithms

A rather simple to understand algorithm is the *alpha algorithm* which scans the process log for relationships between activities (e.g. causal dependency, parallelism, etc.) [14]. One weakness is that the alpha algorithm cannot handle noise in the data very well. This is why other algorithms have been developed on top the alpha algorithm, such as the *heuristics miner* [15] that takes frequencies into account and can therefore filter infrequent behavior. Another algorithm that is based on the alpha algorithm is the *genetic miner* [16, 17]. This iterative algorithm creates many different process models during execution and keeps the ones that fit the log best. Even though it is interesting to use the genetic miner algorithm to create a model from repaired log including parameters such as fitness, it is not well suited for this purpose because the algorithm is non-deterministic and requires much input data. A sufficient amount of input data cannot be guaranteed in the industry data that is used in this thesis. The industry data is described in more detail in chapter 5.

The *inductive miner* guarantees certain behavior, i.e. no deadlocks. This is due to the graph structure that the inductive miner uses in the background [18]. In particular, the process is represented as a process tree first and transformed to a Petri net in a later step during the execution of the inductive miner algorithm. Analyses regarding soundness and behavior of a process model is discussed in more detail in the following section 2.2.

The results of the discovery algorithms have different characteristics. The heuristics miner can discover many process patterns while the inductive algorithms in turn show good results in terms of scalability, fitness to the log and precision [19]. In order to be in line with the technique proposed by [20], we use *integer linear programming (ILP)* [21] to discovery sub-process models during the model repair procedure. The ILP miner combines non-local patterns recognition and structural properties guarantee. Once a process model has been discovered, can be represented graphically and logically in different ways.

2.2 Model Representation

There are multiple options for representation of a process model. In this section, the focus is on those types that do not only serve as a graphical illustration but allow for workflow analysis. Possible characteristics to analyze are structural and behavioral correctness of a process model.

2.2.1 Petri Net

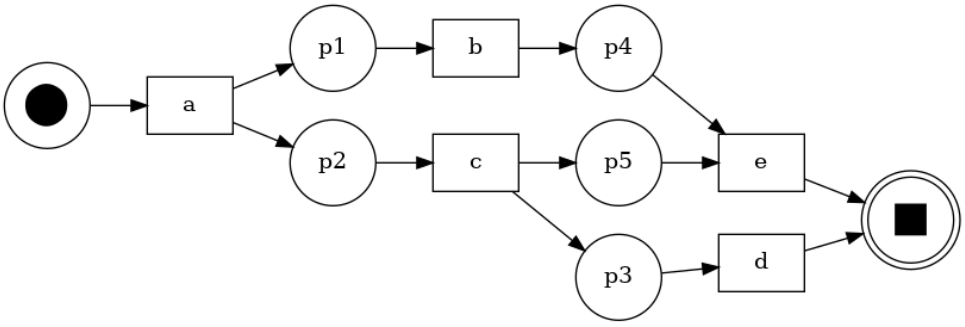
One widely used process model representation is the Petri net. An example is shown in figure 2.1. Petri nets consist of places and transitions with places representing states and transitions representing activities [22, 23, 9, 24]. Places are depicted as circles while transitions are squares or rectangles. Arcs (depicted as arrows) connect places with transitions, but never with other another place. The same is true vice-versa for transitions, that can be connected with places but must not directly precede or follow a transition. One special type of transition is a *silent transition*. It is added to a Petri net if there is the need to execute a transition in order to get to a desired state of execution. Typical cases when silent transitions are added are to maintain formal correctness in a process model, to make a certain transition optional as a silent transition can be added as an alternative path to a certain transition, or to add a loop. Silent transitions can be considered behavior in a Petri net that is for some reason required but does not reflect behavior in the actual, underlying process. Therefore a silent transition has no label because it has no semantic meaning as a regular transition has. A silent transition is typically displayed as a black box without any label or with the Greek letter τ.

Petri nets provide not only a graphic illustration for a process but they are also a mathematical representation that can be used to calculate certain characteristics regarding safe and correct worfklow execution [25]. For this purpose, a Petri net can be replayed which means that a token is set to a place (*source place* that has only outgoing arcs but no ingoing ones and is incrementally moved through the process. A token represents a case that runs through the process, e.g. a patient that gets treatment at a hospital. In a Petri net, a token is represented as a black circle in a place. The token is shifted from current place(s) to the next place(s) by a transition until it reaches the *sink place* that has inverse properties as the source place, i.e. there are ingoing no outgoing arcs. Note that a transition can only execute (also referred to as *fire*) when all ingoing places are marked with a sufficient amount of tokens (usually one token is sufficient). If a Petri net has a source and a sink place and additionally each transition is part of a sequence of transitions (path) that moves a token from the initial state (source place) to the final state (sink place), then this Petri net is a workflow net [26]. The workflow net is designed as a representation of one defined workflow behavior [27]. This defined behavior can be be replayed by a different cases which run through the exact same process specification. In this way it can be checked if there are discrepancies between a intended workflow and cases that represent the underlying process.

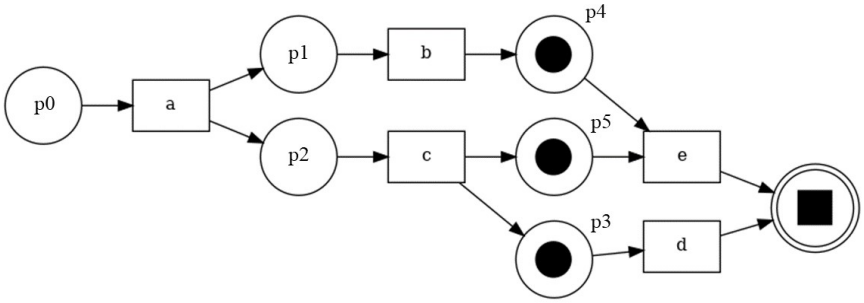
The workflow net guarantees structural soundness but not behavioral soundness. This is displayed in figure 2.3. Note that the marking in figure 2.3a is a token (circle) in the source place and a square in the sink place. The Petri net fulfills the requirements for a workflow net but issues might occur during execution. For instance, if transition a fires and then c fires, transition d is activated and can fire as well which would put a token in the sink node. This means that one token would be left in place *p1* - after the firing sequence $<a, b>$. The marking depicted in figure 2.3b is reached after b fired as well. Nevertheless, there is still a token left because either c or d fires (because there is a token in all ingoing places). In other words, if the marking in figure 2.3b is reached, places *p4, p5* and *p3* carry a token which implies that both c and d are activated but only one of them can fire because the target place can only carry one token. The result is that either one token is left in *p3* when e fires or tokens are left in each *p4* and *p5* when d fires. The structural issue is that transition c opens a parallel workflow that is not properly closed and the final node (sink node) is an exclusive join because the sink node either gets a token from d or e but not both. This remaining token is a problem because it could mean that the output produced by a task is not correctly processed or other inconsistencies can occur in the process execution.

Another type of structural error that can occur during execution of a Petri net is when a token is missing [28]. This can happen when an exclusive split precedes a parallel join, i.e. the parallel join transition would expect tokens in all of its ingoing places which can not be the case because there has been an exclusive split before. This situation is called *deadlock*. Even if a process model is constructed in a way that deadlocks and remaining tokens are ruled out, there is still a situation of an infinite loop that might occur. This is referred to as *livelock* and hinders a process from proper termination. It is illustrated in figure 2.4 with the silent transition between *p1* and *p2* being the loop back to enable transition b again. If it the number of how many times the loop can be entered is not somehow defined, it might be infinitely. Even though a livelock can be considered a less severe problem as a deadlock, it might still have inconvenient consequences if it is overlooked or included on purpose but without a definition of how many times the loop body is allowed to be entered or based on which variable the decision to enter the loop is made and how this variable changes during execution. It might cause troubles when possible paths are automatically computed.

A process that guarantees proper termination and the absence of deadlocks and livelocks is sound. As mentioned above in subsection 2.1.1, process models discovered by the inductive miner



(a) Initial marking



(b) Marking with remaining token

Figure 2.3: Workflow net with remaining token

algorithm are sound by construction. This is ensured by creating a process tree as a prior step to building a Petri net or other model type. As opposed to structural or behavioral inefficiencies, an (possibly) infinite loop can be represented in a process tree as well. Process trees are addressed in the following subsection.

2.2.2 Process Tree

As the name suggests, process trees are a way to express a process model in the form of a tree graph. Therefore a process tree is a directed, connected graph without cycles by definition [29]. Each node is either a branch node or a leave node. Branch nodes are operators while leave nodes are the process activities. There are five types of operators that can occur, namely sequence (depicted as $\rightarrow$), exclusive or ($\times$), inclusive or ($\vee$), parallelism ($\wedge$) and loop ($\circlearrowleft$). The root node is always a sequence operator. The process tree is traversed in a depth-first-search to create a Petri net model from it. The order of a node's children matters for the loop and the sequence operator. For a sequence, the order of the activities is from left to right [30]. For a loop, the left branch is done anyways at least once and the right branch is the loop back. Based on these rules, the Petri net illustrated in the right-hand part of figure 2.1 can easily be built from the process tree in figure 2.5. The process tree itself has been discovered based on the log display in the left-hand part of figure 2.1

One advantage is that the process tree guarantees a sound process model. This is thanks to the block-structure [18]. The output of the inductive miner can be a rather complex Petri net that uses many silent transitions. The downside is that these models tend to be harder to read for humans as they have to incorporate parts that are there just to fulfill the soundness guarantee that is already in the underlying process tree. What is more, the process tree representation allows to measure similarity of two process models by calculating the edit distance between two

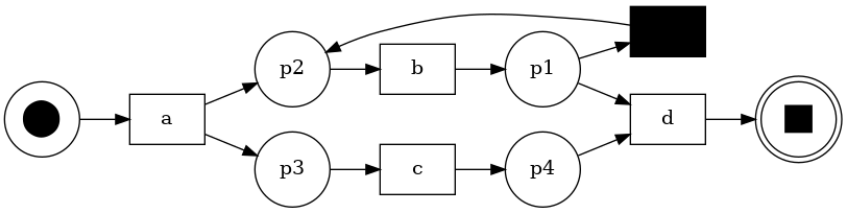


Figure 2.4: Workflow net with a livelock

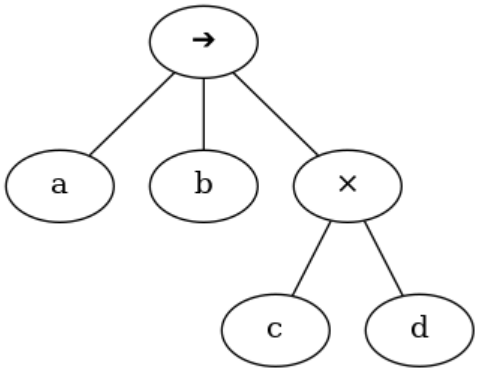


Figure 2.5: Example process tree discovered from the log from figure 2.1

process trees [30].

Other means to illustrate a process model such as Business Process Modelling Notation (BPMN) [8] or directly follows graphs (DFG) are not described in this section as they are not further used in the concept and application of the model repair tasks performed in this thesis. As opposed to Petri nets and process trees, BPMN and DFG figures focus on the graphical presentation of a process to a human observer more than to calculate termination or other process qualities [31]. Hereby BPMN models point out workflow gates such as parallelism, exclusive choice or inclusive choice while the focus of a directly follows graphic usually to depict which activity follows which other activities and how frequent this relationship appears in a process.

2.3 Measures for Model Quality

As indicated in the introduction of process discovery algorithms, these algorithms have an impact on the quality of the model. Model quality can be measured regarding the conformance of a model to log data or vice-versa, regarding the overall structure of a process model (e.g. are there redundant parts that do not add any sensible syntax to the model) or a as a comparison to a reference model. In the scope of this thesis, the five quality dimensions are pointed out to evaluate the characteristics of a model, namely *fitness, precision, generalization, simplicity* and *similarity.*

Fitness is a central metric for the alignment between a model and real processes [32]. Fitness can be determined by attempting to replicate the actual process flow using the target process model. This can be achieved using Petri nets and their associated tokens. An introduction to Petri net replay is provided in section 2.2.1. Fitness describes whether and to what extent a model can represent a process flow. If activities occur in the process log in different location than in the model or not at all, it reduces the fitness. There are different ways how to calculate the degree of fitness. Robust techniques that do not rely on the quality

of log data, identify the discrepancies in form of alignments [33]. Alignments represent the exact deviations between log and model including the location in the model. Nevertheless, simpler approaches such as token-based replay are still in the focus of research and keep up with alignment-based fitness calculation regarding speed and quality of diagnostics [34]. The implementation used for this thesis applies token-based replay. Fitness is just one of several metrics used to calculate the alignment between a model and a process log. However, fitness can be considered the primary metric since a model that does not include the process flow to be compared is unsuitable for conformance checking [35].

Precision is another metric and can be seen as the "inverse" of fitness. Precision indicates the ratio of the possible behavior in the model to the behavior observed in the log. The requirement for a model to have good precision prevents the formation of so-called "flower models" that represent a multitude of possible behaviors but become too general and allow behavior that does not correspond to the actual process. In other words, precision pursues the goal to not underfit the data, i.e. to keep it close to the process data. Since there is possibly unlimited behavior in models (due to loops), precision can only be estimated. Like for measuring fitness, there are different ways to estimate model precision. Simple approaches replay log data on the model and measure the amount of unused parts of the model [36, 37]. Another technique aligns an event log and a process model, enabling more accurate precision measurement, and then measures precision based on the identified deviations [38]. Since the latter approach is only advisable for sound process models, we choose a simpler approach. This mere count of unused model parts assumes that the log data fits the model which is not necessarily the case. On the other hand it gives an idea of a model's precision and is less computationally heavy than computing deviations.

Generalization is the quality dimension that aims to avoid overfitting the model to the data [7]. Like precision, generalization focuses solely on the correlation between the event log and the process model [33]. Nonetheless, the event log represents only a fraction of all the potential behaviors allowed by the model. Thus, the concept of generalization comes into play to determine whether the process model is not overly tailored to the behavior observed in the event log and accurately describes the entire intended process flow. Another aspect of generalization is the likelihood that the process model can effectively capture behaviors that have not been observed yet [7]. Assessing generalization in this thesis is done based on the measure for the notion of generalization introduced in [39]. This measure considers the frequency of visits to specific states and the number of distinct activities observed in each state. When a state is visited frequently and only a few activities are observed, the chances of encountering a new activity during the next visit are low which improves generalization. Conversely, the more parts of a model that are infrequently visited, the worse generalization becomes.

Simplicity aims to reflect the process log by building a model with minimal structure [40]. In the context of Petri net discovery, this means using as few places, transitions and arcs as possible. Simplicity is a characteristic of the sole process model and is - unlike fitness, precision and generalization - independent from comparison to any log data. Simplicity counts the number of causal relations of the model using the cardinality of the input and output subsets of the causal matrix. This means that the number of enabled activities is counted before and after the execution of an activity. In other words, we consider the pre-conditions and post-situation of each transition execution. If there are fewer transitions necessary to be fired in a model in order to get the same result as in a reference model, then the first model is simpler.

Similarity is used to measure how strong a repaired model differs from the original one. For model repair, similarity is worth being considered because we assume that the original model contains valuable information about the process that must not be neglected by overwriting the model. Hence one goal of model repair is to keep the repaired model as similar to the original as possible while the required adaptions are made. One possible way to assess similarity is to measure the distance between two models. This measure of similarity is used in [30] where a reference model is updated and an indicator for structural similarity between the reference model and a candidate model is required. In this thesis, it would be sufficient to consider structural similarity instead of behavioral similarity since we want to compare two models independent from event log. For this purpose, the edit distance between the process trees of the models can be calculated as presented by Zhang et al. [41]. However, a process model can only be transformed into a process tree if it is sound. This is a limitation for the models discussed in this thesis as only those models discovered by inductive miner guarantee soundness. The avoid-flower models are designed to maintain soundness while all other repair procedures do not explicitly target soundness. Therefore, we simply use the number of changes during model repair to assess similarity. In other words, we count how many change steps are carried out to transform the original model into model a repaired model.

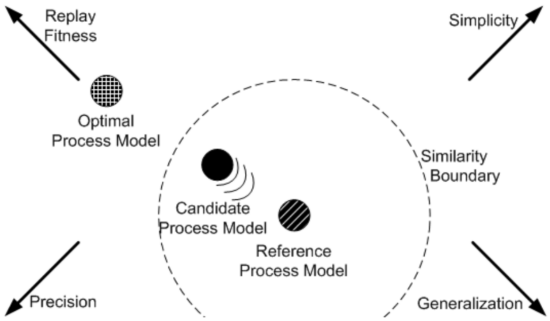


Figure 2.6: Quality dimensions of process model discovery including similarity boundary [30]

Figure 2.6 shows that the quality dimensions as driving forces of process discovery are partly contradictory and need to be balanced. The fifth dimension similarity adds a boundary to the model that determines to which degree an adapted model (candidate model) is allowed to differ from the (original) reference model. Again, this emphasizes that model similarity is not one of the quality dimension considered in the phase of model discovery but it is used when an exiting model is adapted.

After the introduction of concepts and means to create process models, how they can be represented and how their quality can be measured, the focus in the next section is on model repair. The notion of model repair as part of process mining is presented in the following chapter.

3 Model Repair

Model repair is typically categorized as a part of process enhancement which itself is one of the three main phases of process mining. Process enhancement aims to create models that fulfill conformance to reality, compliance to laws and regulations and support prediction-making [13]. It starts with a process model that is discovered from data or designed by hand and requires adaptions. Model repair is the procedure which aims to adapt a process model to reflect recorded reality. Model repair can be seen as applied process improvement. Besides process improvement, model repair serves other purposes, such as improving conformance checking diagnostics, monitoring process evolution over time or increasing process customization.

3.1 Alignments

When it comes to model repair, it is required to find out in which way a model must be adapted to attain a more desirable version of it. As already mentioned above in section 2.3, one possible way to express discrepancies between log data and a model are **alignments**. Alignments are an output of conformance checking. This goes along with the idea of process enhancement as a subsequent step of conformance checking. In other words, alignments are the output of conformance checking and serve as basis for the process enhancement procedure, i.e. model repair. The concept of alignments has been introduced and used to obtain a comprehensive view on different types of conformance checking [39, 42, 43]. When we want to identify alignments, we try to replay an event log on a process model. We relate "moves" in the log to "moves" in the model in order to establish an alignment between model and log. The output of alignments analysis is a list of moves referred to as γ. Each move is a pair with each element either being a process step or "no move". It is not allowed that both elements of a pair are no move. No move represents a missing process step in the execution sequence. Formally, a move is written as $(s_1, s_2) \in (S_\perp \times S_\perp) \backslash (\perp, \perp)$ with S being the set of possible execution steps, $\perp$ the representation of no move and the set $S_\perp$ being the union of possible execution steps and no move, formally $S\perp = S \cup \perp$. There are three types of moves:

- Log move: an activity is in the log but not in in the same location of the model. Formally: (s_1, s_2) is a move in log if $s_1 \in S$ and $s_2 = \perp$
- Model move: activity is in the model but not in the same location of the log execution sequence. Formally: (s_1, s_2) is a move in process if $s_1 = \perp$ and $s_2 \in S$
- Synchronous move: activity are in model and alternative log in the same location. Formally: (s_1, s_2) is a synchronous move if $s_1 \in S$ and $s_2 \in S$.

Synchronous moves do not trigger any repair action since it indicates conformance between log and model for the process step it refers to. On the contrary, model moves and log moves provide a list of which changes would be required in order to bring the model into accordance with the log.

Replay of a log on a process model is not trivial. Multiple alignments are possible to fit one trace to a model. An alignment is optimal when the number of deviations for visible transitions between a trace and a model, is minimal [39]. The conformance or non-conformance of a sequence of steps depends on which path is taken during the execution of the model. The goal is to find a path in a graph from a given source node to any node in a target set. There are different algorithms to solve this problem. The implementation used for this thesis applies the Dijkstra

Less Memory algorithm [44, 43]. For this algorithm it is possible to assign costs to moves or types of moves. If certain moves should be avoided, high costs are assigned to this type of moves. In the context of model repair we assign higher costs to log moves and model moves than to synchronous moves. Log moves and model moves potentially trigger model repair action which should only be carried out if it is necessary. If there is a way to replay a trace using synchronous moves instead of log moves and/or model moves, then there is no need for repair. If the costs for synchronous moves are not lower than for log and model moves, the output of alignments analysis could possibly include log moves or model moves and hence repair would be carried out even though the log would already be compliant with the model.

3.2 Repair Technique

How alignments can be used for model repair is presented in the approach by by Fahland and van der Aalst [20]. They demonstrate model repair using recorded process executions from the past. The motivation is to adjust the process model to reality. This means that the process model to start with has possibly been drafted by the process owner or someone else who has knowledge about how the process is supposed to be executed. But this does not mean that the intended process matches the actual, real process execution. The goal is to include behavior from the log to the model. Concrete execution steps that are not yet in the model and are to be added, are pointed out in the alignments as log moves. Parts of the model that are not in the logs and should be made optional are represented as model moves.

The main steps of the repair technique by Fahland and van der Aalst are

1. *Add loops*, which tries to find repeated behavior that is to be included in the form of a loop
2. *Repair subprocess*, which groups behavior that is to be added at the same place in the existing model, and
3. *Remove unused parts* which excludes parts that make sense according to the repair technique but are too infrequent to actually improve the model. This is done to keep the model close to real behavior, especially when the repair logic is tested with log data.

Note that before identifying loops and subprocesses, log traces are grouped based on the alignment moves that would be required to make the trace replayable. Furthermore, the location in the model where additional behavior is to be added, is reduced by identifying overlapping places that multiple loops or subprocesses have in common.

An example of model repair based on alignments is shown in figure 3.1. Note that the original model in 3.1a is the same as the Petri net in figure 2.1. Let us assume that we want to compare the original model in figure 3.1a to the following process log: $\sigma_1 = <a, b, e>$, $\sigma_2 = <a, c>$, $\sigma_3 = <a, d>$. Table 3.1 shows the alignments between model and log per trace. For the first trace $\sigma_1 = <a, b, e>$, there is a log move for activity e: *(e, »)* since e is not in the original model. There is a model move for activity c: *(», c)* because it is not in the log but is a mandatory process step in the model. Alternatively, a model move for d would have the same effect and align the trace to the model. In the other two traces, there is a model move for b. The lists of alignments per trace are referred to as γ_1, γ_2 and γ_3, i.e. the alignments for trace σ_1 are $\gamma_1 =$ *(a, a), (b, b), (e, ») and (», c).*

These alignments can serve as basis for model repair. Figure 3.1b shows the repaired model with the added parts colored in light blue. Activity b is now optional so σ_2 and σ_3 can be replayed. Furthermore, transition e and one silent transition have been added. The silent transition serves as a third path between *p2* and the sink place. As a result, σ_1 is now replayable as well. However, the repaired model as it is displayed in figure 3.1b is not an optimal solution to make the log

Table 3.1: Alignments of log and model

$\gamma_1 =$

a	b	e	»
a	b	»	c

$\gamma_2 =$

a	»	c
a	b	c

$\gamma_3 =$

a	»	d
a	b	d

replayable. It would be more advantageous if e is added as a third path instead of the silent transition between *p2* and the sink place because it would still allow for replay of the full log with one transition fewer. This issue is addressed in [45]. The result of using additional logic to repair log moves in decision structures results in a more elegant solution to this problem. It is depicted in sub-figure 3.1c. Note that the additional repair technique for choice structures is only sensible if the choice between either c, d or e is compliant with every trace that needs to be repaired. Clearly, if the model is required to replay another trace $\sigma_4 = <a, b, e, d>$, 3.1b is the only model that incorporates this behavior as only here c can be executed after e has already been executed.

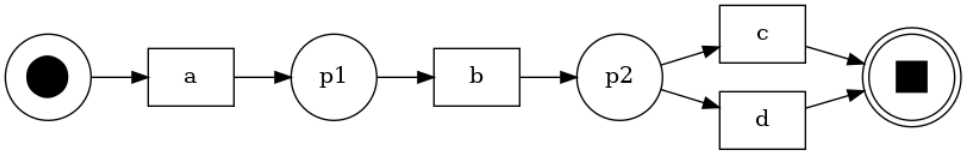
The model in 3.1b has been repaired based on each move separately. This means that log move *(e, »)* and model move *(», c)* are repaired independently even though e "replaces" c in this trace. There are some more possible issues: the way the transition for activity e is added, might cause problems since it is a loop without an explicit loop body as *p2* is both the preceding and subsequent place of transition e.

Another question that arises is if the silent transition between *p1* and *p2* is an optimal repair step. Otherwise, it might be better to exclude b from the model since b is not present in the majority of traces that the model is compared with, here in two of three traces. This depends on the way the original model was created. If it was drafted by a domain expert, there will be a good reason why b is in the model and why it is in the exact location between *p1* and *p2*. If the model has been discovered by using the most frequent variants of a log that is available, the question if b should be removed, might be worth discussion.

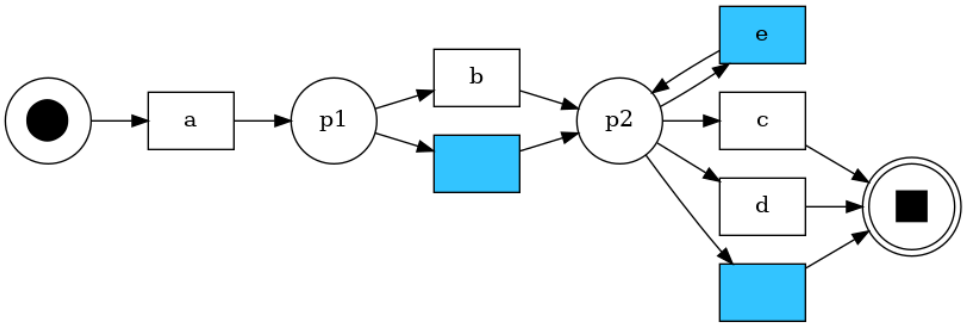
3.2.1 Avoid Harmful/Wrong Repair Steps

The possible issues described just above point out the importance of the (original) model to start with and the trace that is used to identify alignments. Of course, thorough data cleansing and preparation is necessary to ensure a sensible outcome of the model repair procedure. Furthermore, it is important to have a clear idea how the model to start with is created and whether it should be possible to adapt this model not only by adding transitions but also by removing ones. The technique of how additional parts are added must be considered as well. It might be undesired to add a transition in the way e is added to the repaired model since there is no option to check if the loop should be entered or not during execution. This creates a flower model and might cause unintended behavior on execution of this model. It can therefore be advantageous to apply a different method to integrate a subprocess into a model. In this case the subprocess consists only of transition e. The issue is that the subprocesses starts and ends at the same place, namely *p2*. An alternative way to integrate short subprocesses is presented in section 4.2.

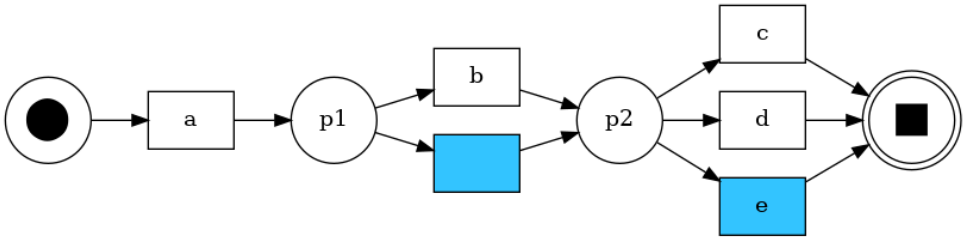
During the procedure of model repair, one has to keep in mind that the model might be falsified or otherwise edited in an undesired way if parts are automatically added. The question how to avoid this can be answered by the nature of the data that is used to find repair steps. While the



(a) Original model



(b) Repaired model with additional behavior using technique by [20]



(c) Repaired model with additional choice structure repair [45]

Figure 3.1: Comparison between original model and repaired versions

original model represents a planned process, the log data is real tracked process data, i.e. actual behavior. It is needed to ensure that reliable data that is firstly conform to rules and regulations and secondly represents the actual procedures of a business or manufacturing process, serves as the base for process enhancement. If this condition is fulfilled, only these steps that have already been at least once tracked and analogous to desired real world steps, can be included into the model. This is relevant because repair steps that are not sensible can be harmful and lower the benefit that model repair offers.

3.2.2 Model Repair vs. Process Discovery

For purpose of this paper, it is important to distinguish process discovery from model repair. Model repair is guided by two forces: one is to improve the model by bringing the model closer to the variants with better KPI values; the other one is to keep the repaired model as close to the original model as possible [46]. If the similarity to the original model is neglected, model repair can be done by just applying discovery algorithms to the repaired log [20]. To point out the difference between a discovered and a repaired model, the results of both approaches are compared in this paper. The repaired model should naturally remain more similar to the original model, while it is not necessarily expected to have better scores in precision, generalization, simplicity and most importantly fitness. Of course these measure depend on the type of discovery algorithm as well [19].

3.3 Key Performance Indicators

As mentioned above, the aim of this thesis is to use Key Performance Indicators (KPIs) for model repair to improve model performance. In this section an overview of the use of KPIs for process mining in the context of manufacturing processes is given.

KPIs are commonly used to signal whether managerial action for improvement is required [47]. Although, in some cases KPIs remain unrecognized. This means the identification and correct use is not always obvious. The advent of manufacturing execution systems (MES) as part of an automation system in manufacturing industry has raised the need to integrate components (Enterprise Resource Planning and other software) but also aroused interest in a standard for KPIs [5]. Typical indicators can include performance, quality, cost, delivery, safety, morale and environment. These form part of the TPM (Total Productivity Management) method [5].

Linderberg et al. proposes a list of KPIs based on literature research done in various manufacturing sectors [48]. Table 3.2 consists of some simple KPIs in the list by Lindberg et al. and other KPIs that are used for business process improvement [49].

Table 3.2: Example KPIs based on [48]

Example KPI (Formula)	**Comment**
$\frac{\text{Energy output}}{\text{Energy input}}$	This expresses the energy efficiency of equipment [50]
$\frac{\text{Energy input}}{\text{Produced output}}$	Indicates energy consumption per physical output (unit). Can be used to compare energy performance across factories, companies, etc. [50]
Overall Equipment Effectiveness (OEE)	OEE is a comprehensive analysis tool for equipment performance. It can be split into parts that deal with downtime, speed losses, quality losses etc. [51]
$\frac{\Sigma \text{ throughput times of all parts}}{\text{number of parts}}$	This is the average throughput time [52]. Throughput time is a typical measure in process mining [49].
$\frac{\text{Throughput rate}}{\text{Average Inventory}}$	Inventory performance is strongly influenced by input material and maintenance performance (spare parts) [51]

Although KPIs are defined to be standardized and the selected KPIs presented in this paper are simple, they undergo changes such as an increased focus on sustainability and eco-friendliness [50]. This puts special attention on indicators that regard energy consumption and efficient use of resources.

Bauer et al. shows that the combined use of planned and actual execution times can improve performance [53]. Furthermore, correction of scheduled production processes is mentioned as

a means to overcome insufficient process performance. May et al. introduces a framework for KPI-based monitoring of energy efficiency in manufacturing industry. The aim is to provide a method for companies to create tailored indicators in terms of energy consumption. Arinez et al. describes how a discrete event simulation model helps to improve decision-making [54]. The model is built on input data such as KPIs and the output data serves as source for bench-marking of the overall manufacturing system. These examples show that not only KPIs are used in an increasingly customized way but also that performance monitoring with KPIs is tightly coupled to predefined processes in organizations. Adapting these processes can be part of measures to reach performance goals.

3.3.1 Standards for Process Improvement

A standard that focusing on KPIs in MES is ISO-22400 that is published by the International Organization for Standardization (ISO) [4]. The activities to use KPIs to improve processes of an enterprise is referred to as Manufacturing Operations Management (MOM) in ISO-22400. MOM is integrated into the functional hierarchy model of a manufacturing enterprise [4]. MOM and MES are used synonymous in the ISO standard. MOM comprises four categories (production, maintenance, quality, inventory), each of which is again split into eight activities. These details are defined in the ISO standard [4]. It aims to facilitate the specification and procurement of integrated systems and standardizes productivity tools to be applied across different MES applications. Thresholds are mentioned as means for improving a process in a particular company, along with warning and action limits. KPIs can also contribute to decision making in business planning and logistics. KPIs in these areas are often related to external factors such as economic, business, logistic and financial factors.

Six Sigma aims to improve business performance based on statistical analysis [2]. The key methodology to deal with business activities is the so-called DMAIC methodology. DMAIC is an acronym of the five phases it comprises: define, measure, analyze, improve and control. The phases *define* and *measure* are described to be the most challenging when applying Six Sigma methodology [55]. In particular, lacking process orientation is seen as one of the causes for problems for determination of KPIs in the measure phase. Hence it is vital to include not only business process expertise but also domain knowledge to problem definition and performance measurement.

An example for the application of Six Sigma in the context of business process optimization has been published by Ray and John [56]. They propose a combination of Six Sigma and lean methodology (which seeks to minimize non-value-adding activities [57]). Discussions and brainstorming among involved team members were the means to come up with an improved process in the study presented by Ray and John. An automated approach to process improvement by adding or removing tasks is missing in this case. Methodologies for approaches to automated business process improvement are influenced by Six Sigma [6][1]. Six Sigma is a widely used concept that provides a basis for data collection, processing and evaluation.

3.3.2 Overall Equipment Effectiveness

As illustrated in table 3.2, time (e.g. throughput time) is - among cost, quality and flexibility - one of the “classic” indicators of business process performance [49]. Furthermore, the *effective throughput*, i.e. the number of conforming parts produced in a time period, is mentioned as one of the most relevant performance measures in production processes [58]. Another example is presented by Ellis et al. who reduced costs of material flow due to model-supported process analysis [59]. More particularly, they focused on handling time as well as logistics figures, such as storage capacity and assembly line locations. This shows that there is a bridge between

KPIs typically used in manufacturing industry and measurements that are common in process analyses.

It stands to reason, that throughput time as a KPI is probably not best suitable for manufacturing processes because optimization based time as only target variable might contradict other - more important - goals such as work safety. The aim is to optimize a process with the help of KPIs of process instances. Hence it is desirable to consider KPIs that comprises multiple aspects that contribute performance and quality of a process. One example of a comprehensive KPI is **Overall Equipment Effectiveness (OEE)**. It is an important performance indicator that is widely used in manufacturing industry. OEE - which is also mentioned in table 3.2 - is not always defined and used consistently in literature. For example, in [58] OEE is used synonymous to effective throughput. All in all, OEE can be described as a comprehensive analysis tool for equipment performance with a focus on processing time and quantity of good output.

For this thesis, OEE has been identified as the most important KPI available in the data that is analyzed in the scope of this thesis. This decision has been made along with the provider of the manufacturing process data. A thorough description of the data can be found in chapter 5. The way how OEE is calculated her is based on the calculation of OEE as presented in VDMA standard sheet 66412. This standard sheet contains business performance indicators in the field of manufacturing controlling and particularly manufacturing execution systems. The three aspects that make up OEE are *availability*, *performance* and *quality*.

In the context of this thesis, three aspects of OEE are calculated as follows:

$$\textbf{Availability}: \quad \frac{\text{Processing duration excluding disturbance}}{\text{Sum of durations}}$$

$$\textbf{Performance}: \quad \frac{\text{Processing duration excluding disturbance}}{\text{Planned processing duration}}$$

$$\textbf{Quality}: \quad \frac{\text{Good output}}{\text{Total output}}$$

For calculation of availability, the sum of durations (denominator) is the sum of (1) processing, (2) set-up, and (3) disturbance duration. The nominator is the processing processing duration reduced by disturbance duration. Performance is calculated as with the same nominator as for availability. The Planned processing duration (denominator) is determined based on the produced amount which means that it is the sum of the planned durations of the output quantity. Total output, that is the denominator in quality calculation, is the sum of good output, rework and waste quantity.

The use of OEE allows to measure the impact of multiple aspects such as output quality, processing duration and efficiency, by using only one indicator. This makes it easier to identify processes where the characteristics of process instances have a statistical impact on this selected KPI. Based on statistical analysis, those processes where the discrepancies between process instances make a difference, can be selected for KPI-based model repair.

3.4 Related Work

This section presents work that has been done in the field of model repair with a focus on KPI-based approaches. One topic of particular interest is data from manufacturing processes. As

indicated in the beginning of chapter 3, model repair is based on an existing model that is to be adapted. Various approaches to edit a model with different goals have been developed and applied.

An overview of selected related work on approaches to model repair is given in table 3.4. The work is sorted in characteristics *intention*, *goal* and *data* with two options each. The last row says whether extensions to an existing repair technique is proposed as this thesis does. Note that *intention* refers to the general approach how the structure of a model should be built, i.e. whether it is built from scratch (discovery) or an existing model is adapted. On the contrary, *goal* refers to the motivation or moving factor that shall be achieved.

Often times, the general intention of model repair is what makes the difference between the approaches. Table 3.4 shows that related work can be split into discovery of a new model and, on the other hand, repairing an existing model. Moreover, model repair can pursue different goals. Examples are to improve performance, build a model based on variants created in the past, or automatically correct errors to reach model soundness. Table 3.4 distinguishes between model repair as KPI-based optimization and adjusting the model to a changed process. As a result of different goals, the repair procedures differ as well. Li et al. [46] focuses on creating a new reference model from past process changes with the help of machine learning. In particular, the algorithm performs mining of process variants. This approach provides flexibility regarding how much the reference model differs from the original one. A downside is that it becomes heavily computation-intensive when working with large process models.

Process changes due to organizational reason is the basis of the work by Polyvyanyy et al. [60]. Similar to [20], a function to align costs to repair steps and choose an optimal strategy based in this is introduced. Nevertheless, there is no connection to KPIs in this approach. An earlier paper shows how a workflow can be specified and changed in an open instance, i.e. at run-time [61]. In another publication, a model repair technique that emphasizes behavioral correctness is introduced [62]. It is based on local mutations. Work to avoid deadlocks has been done by Lohmann [63]. The minimal edit distance is used as measure between a given defective process and synthesized correct process.

Work by Schunselaar points out how existing process models can be combined to achieve an optimal process model based on the configuration of KPIs [64]. Hence, this approach combines KPI-based improvement and adjusting the model due to a change in the represented process. Although the result is an improved process model that fits the needs of a flexible work environment, a downside is that a set of configurable process models is required. This means that the outcome of this procedure depends on domain knowledge and sufficient historical process data. What is more, the concept is tailored for the use in organizations like municipalities with a focus on administrative processes.

Process model repair with application of the technique by Fahland and van der Aalst [20] is presented in [65, 66, 45]. While Saelim et al. [65] emphasizes the identification of loop structures in Petri nets, Qi et al. [45] proposes an adaption to Fahland and Van der Aalst's technique by using extended alignments to repair choice structures. The aim is to avoid an output process as depicted in figure 3.1 and obtain a more elegant process model. Both papers present their findings based on case study data form the health care sector. Note that work by Qi et al. is not intended to optimize a KPI or fit a model to a changed process. Rather the aim is to extend the repair technique proposed by Dees et al. [1]. Armas Cervantes et al. [66] implemented a tool that visually shows discrepancies between log and model to the user and lets them decide which parts to repair.

It can be seen that work with manufacturing data is rare and papers that deal with such data do not apply KPI-based repair of an existing model. For instance, work by Lugaresi et al. [67] deals with manufacturing process data but focuses on model discovery with an approach to

combine discovery algorithms. Apart from work by Lugaresi et al. and this thesis, other work does not include model repair on manufacturing process data.

Further work by Shahzadi et al. [68] points out alignments as features for automated process model repair with the goal to improve model performance. Although some aspects are very similar to the technique applied in this thesis, Shahzadi et al. perform model repair to mend missing or incorrect log data and improve the process based on the improved data. Repairing event logs regarding missing data with the purpose of correct documentation to ensure compliance and support further administrative tasks is presented in [69]. Alignments can serve for conformance checking in a comprehensive way including incorrect routing decisions, incorrectly assigned variable values or issues with unqualified resources [70]. The basis for these analyses is finding alignment between model and log data which has been done specialized by Song et al. [71].

A general approach to include KPI-based process improvement into the Business Process Management (BPM) lifecycle is proposed by Cherni et al. [72]. The idea is to combine KPIs and process mining techniques for process improvement. Frameworks for continuous process observation with KPIs can be applied to identify undesired behavior during process run-time [73, 74, 75]. These frameworks are applied as full monitoring tools that gather information about the process, apply machine learning techniques to assess the influence factors on certain KPI values and display analyses e.g. as decision trees. However, there is no automated process adaption included.

		Li et al. [46]	**Lugaresi et al. [67]**	**Schunselaar [64]**	**Polyvyanyy et al. [60]**	**Gambini et al. [62]**	**Armas Cervantes et al. [66]**	**Dees et al. [1]**	**Saelim et al. [65]**	**Qi et al. [45]**	**Strasser 2024 (This Thesis)**
Intention	Discover new model	X	X	X							
	Repair existing model				X	X	X	X	X	X	X
Goal	Adjust model to changed process	X	X	X	X	X	X				
	KPI-based optimization			X				X	X		X
Data	Administrative domain	X		X	X	X	X	X	X	X	X
	Manufacturing domain		X								X
Adds extensions to model repair based on process data										X	X

Table 3.4: Overview of related work

3.4.1 Research Gap

Research has been done in the field of targeted process model adaption for various purposes as well as on the use of KPIs to monitor and control business processes over time. A combination of these two foci is proposed in the work by Dees et al. [1] since the outcome is a general repair technique that can by applied on any event log. Furthermore, it is relatively easy to comprehend and repeat as the focus is not on configuration or certain variants but only basic process adaption steps are applied such as adding transitions in certain places to a Petri net.

The approach discussed by Dees et al. is named **KPI-based model repair**. It serves as the basis for the work of this thesis. However, the use case of data from real world manufacturing organizations is a novel aspect in the field of model repair. The available data from organizations in manufacturing industry allows to apply existing model repair techniques. Process optimization is an issue in manufacturing as well as it is for administrative tasks. Hence model repair on manufacturing data using an adapted version of alignment-based model repair is a promising way to adjust process models with regards to specific performance indicators.

Since KPIs are available in a wide range in industry (such as manufacturing), new insights can be found when working with broad data sets from industry. This goes along with adaptions to the repair technique in order to obtain more concise process models that are better suited for use in practice. Furthermore, quality measurement as described in section 2.6 can be conducted easier and more comprehensive if the model is built under certain quality constraints.

The literature overview in table 3.4 shows that only few work is done on KPI-based optimization. Most work emphasizes behavioral correctness or fitness to log as target of model adaptions. Furthermore, only one related paper listed in table 3.4 deals with manufacturing data, namely Lugaresi et al. [67]. However, this paper is about model discovery rather than model repair. Other than existing work on KPI-based model repair, this thesis focuses mainly on data from manufacturing industry. The outcome of this thesis is a model repair procedure that automatically adds behavior to Petri net models using an updated version of the procedure by Dees et al. [1]. The update comprises two extensions, namely ***avoid live-locks (avoid flower-models)*** (abbreviated *avoid-flower*) and ***use move including location as feature*** (abbreviated *move-loc*). These extensions are presented in section 4.2. The intention is to provide process owners with a tool to directly adapt a process model based on target KPIs rather than providing an overview in form of a dashboard or similar concepts that require further action by process owners or analysts. The extensions proposed in this thesis are designed to complete the model repair approach tailored to manufacturing data.

4 Concept and Implementation

As mentioned in the chapters above, the motivation of this thesis is to use model repair to achieve improved KPI values. The basis for model repair are alignment moves that indicate discrepancies between a model and process log. In this chapter, the method of KPI-inspired model repair is discussed. The methodology presented here is based on the work of Dees et al. [1] with extensions to achieve more concise, realistic or syntactically correct models. The extensions are addressed in subsection 4.2.

4.1 KPI-inspired Model Repair (Dees et al. [1])

The goal of KPI-inspired model repair is to make processes better by combining theoretical knowledge of KPIs in the context of manufacturing with process improvement techniques. This is done using KPIs to guide automated recommendations for edits in the model. We aim to correlate model moves and log moves for the recorded data to a chosen KPI. This can be done by learning a classification tree which predicts the KPI value (dependent variable) by using the number of moves as independent (input) variable. Each leaf of such a classification tree can be seen as a cluster of traces. The path from the root to a leaf represents a set of rules that tells which activity to skip or execute in order to reach the KPI value stated in the leaf node. The classification tree learns which KPI values are assigned to which trace and predicts if including or ignoring certain moves leads to a better or worse KPI value. All traces that are predicted to have the same set of rules (i.e. set of moves) are grouped into the same cluster. Note that wrongly classified traces are removed. This means that a trace that is predicted to have a desirable KPI value but the actual KPI value is not desirable, is not included into the log cluster. This is done to disregard cases which represent behavior that leads to an unsatisfactory KPI value. Otherwise, if the predicted KPI value and the actual KPI value of a trace are satisfactory, this trace is included to the log clusters so that the discrepancies between this trace and the reference model become a part of the repair procedure. The way how a decision tree is created from alignment moves is described in [76].

In a further step it is necessary to repair and merge the log clusters. Repairing the log clusters means to either exclude activities that are not allowed (due to regulations) or make sure that mandatory activities (e.g. compulsory from the business domain point of view) are included to traces. We iterate over each cluster associated with a classification rule (represented in a leaf node) and repair them as described previously. Furthermore, we repair all deviations linked to behavior that is not correlated to improved KPI values. Afterwards all repaired log clusters are merged into a single event log from which a new optimized model can be built. The repaired event log can be the basis for both model repair or new model discovery. The differences between these approaches are discussed in section 3.2.2.

An example is illustrated as decision tree in figure 4.1 and corresponding repaired model in figure 4.2. Here the KPI is a numeric value that is more desirable if it is higher. One can interpret it as e.g. the quantity of good output. Since this is only a constructed toy example, there is no real meaning of this very process logs as they are just for demonstration. As threshold to distinguish desired and undesired KPI value, the mean value of the KPIs of all input traces is used which is 600. This means that traces will be aligned according to those clusters that have a value larger than 600 as leaf node. It can be seen that if we follow the right path from the root to leaf node A, this is a node that represents a cluster that is included to the repaired log. Leaf node A has a predicted value of 1000. The decision that is represented in the root node is *('»',*

'e') <= 0.5 which means that a model move on event e is **not** included. As the branch from the root node to leaf A answers this decision with *False* and leads to an improved KPI value, we will include the model move on e. There is a double negation here as we see that the absence of the model move on e does not lead to an improved value but the opposite is true. Similarly, the other leaf node with a desirable KPI value, namely leaf node C represents a set of rules and each trace that these rules apply to, will be included to the repaired log. In the case of leaf node C, the rules are not to include the model move on e but to include a log move on w. Leaf node B is not associated with a desirable KPI value hence it is not considered. Even if it would have been considered, the path from the root node to leaf B includes no moves at all which would mean no changes to the original model.

As a result, we have two rule sets that lead to a desirable KPI value. In the clustered log data, over all traces, there is no trace that includes only the model move on e or only the log move on w but not both of them or none of them. In other words, it is not required that the model allows e to be skipped while at the same time w is not part of the model. Therefore, in this situation it is allowed to apply the technique by Qi et al. [45] to obtain a more elegant process model when we want to repair a log move at a decision point.

The repaired model depicted in figure 4.2 is then used to compare it to the original model regarding KPI values of traces that fit to these models. The log data has been split into train and test data. While the train data is the traces that have been used to create the decision tree and to repair the model, the test data is now used to check if the model improved based on data that it has not yet seen. In particular, we filter for those traces from the test data that are replay-able by the original model and those that are replay-able by the repaired model respectively. Then we calculate the mean KPI value for these fitting traces. We also consider the median and standard deviation as there might be outliers.

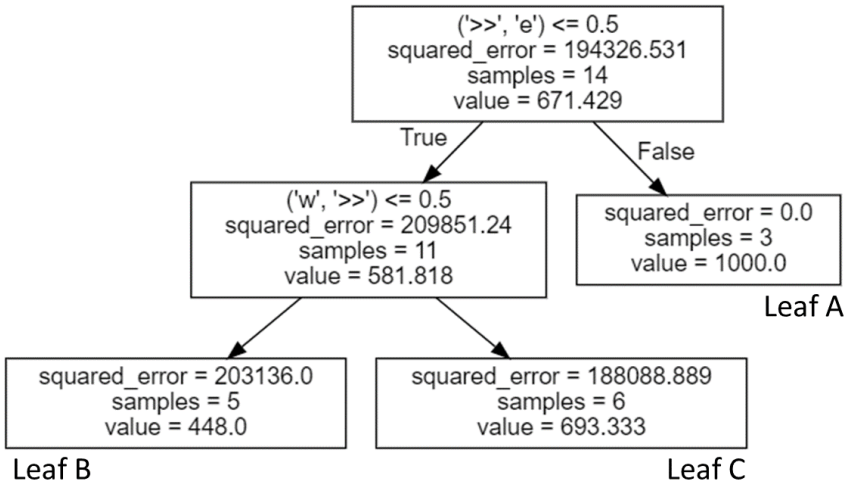


Figure 4.1: Decision tree correlates alignment moves to KPI values following the example by [1]

Examples from real world business cases are discussed in the work by Dees et al. [1]. They point out that apart from better KPI values, it is necessary to create a model that is compliant to rules and regulations and includes activities that are indispensable from a business point of view. What is more, criteria that are specific to process mining need to be fulfilled: the model should not be under-fitting, i.e. allow for more behavior than the log data and the model needs to be sound, i.e. it does not contain deadlocks and guarantees proper execution. For the case studies mentioned, commonly used data mining practices were applied such as to split the data

in training and test data.

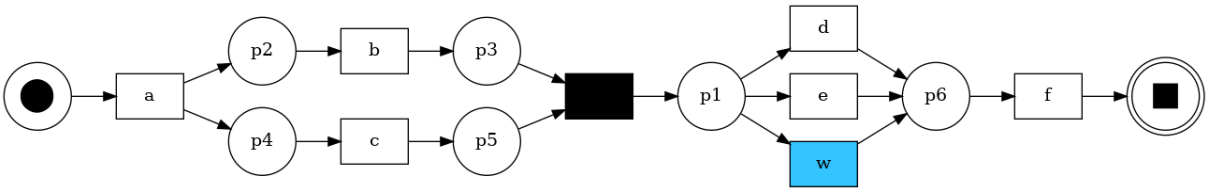


Figure 4.2: Repaired model with increased KPI value following the example by [1]

4.2 New Extensions to Repair Technique

The methodology discussed so far is close to the work by Dees et al. [1]. On testing this methodology, possible extensions and changes were identified that address model soundness issues and fit the methodology better to the application on manufacturing data. Two main adaptions are implemented for the analysis:

4.2.1 Avoid Flower Models (*avoid-flower*)

The way sub-processes are integrated to an existing model in the approach by Dees et al. can cause problems. This has already been pointed out by Qi et al. [45] - see 3.2. However, the method to insert transitions by Qi et al. is only applicable if (1) there is a choice structure and (2) there is no trace that includes the transition to be inserted but needs to skip the next structure. An example where the adaption by Qi et al. does not make sense is depicted in figure 4.3. Unlike in figure 3.1c, it does not make sense to replace the silent transition with the newly inserted transition *b*. While the first condition holds true - there is a decision between *c* and the silent transition when place *p1* is marked - repairing the model by replacing the silent transition by *b* would not represent all logs that are to be added, namely $\sigma_1 = <a, b>$, $\sigma_2 = <a, c>$, $\sigma_3 = <a, b, c>$. Trace σ_3 could not be replayed if there was an exclusive choice between *b* and *c*.

The purpose is to avoid a structure as it is depicted in figure 4.3a since this is a live-lock. The way transition *b* is included to the model might result in an infinite loop. Even though loops can be part of sound Petri nets as long as they are in line with block-structured workflow modelling languages, the process flow depicted in figure 4.3a might cause problems as there is no explicit loop body nor entry and exit points [18]. Hence it is not possible to transform a model with this structure to a process tree.

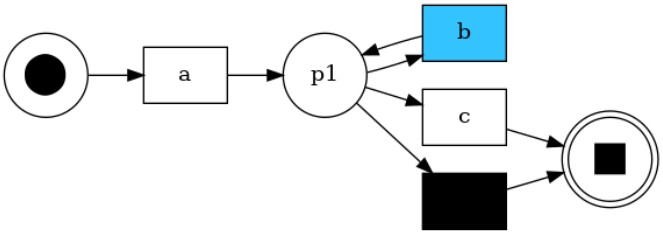
The model repair technique by Fahland and Van der Aalst [20] is prone to create structures like this. They can be avoided by adding the sub-process as a sequence into the existing model and not as a separate sub-process that has the same source and sink place within the existing model. This kind of solution is depicted in figure 4.3b. Not only *b* and a silent transition to make *b* optional are inserted, but also an additional place *p0*. As a result, *b* can be executed before *c* but only once. Other options such as to insert *b* in a well-formed loop would make the model unnecessarily complex. A loop does not reflect the reason why *b* is to be added. It is only necessary to add *b* as a single activity to the model. Hence a loop allows for more behavior than required which results in a reduced precision of this model. As mentioned above, models with bad precision score are called “flower models”. Structures where a transition has the same consecutive and previous place are typical for such models.

A downside of the solution shown in figure 4.3b is that does not allow for *b* to execute multiple times as *b* puts the token from *p0* to *p1*. Although this is probably not a problem in small examples where we only want to add activity *b* once, it might cause problems in larger examples.

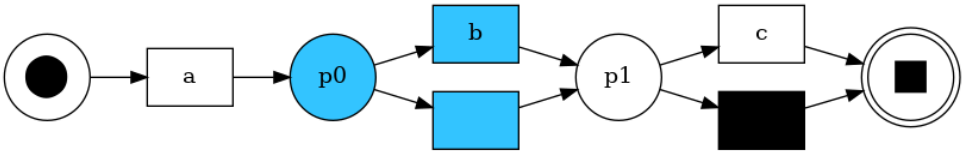
This is even more an issue when multiple activities are to be inserted at the same place. The "avoid flower model" solution adds the activities one after each other causing the model to only allow the transitions to be fired in the order they are added. This does not only lead to a lengthened process model but can also diminish model precision. Again, these problems mainly occur for larger processes with different kind of behavior to be included. For small examples, the avoidance of live-locks (and flower models) can be practicable.

4.2.2 Use Move including Location as Feature (*move-loc*)

As opposed to the technique by Fahland and Van der Aalst [20], we distinguish alignment moves not only by the type (log move or model move) and event they refer to but we also consider the location of the alignment. The location is represented by the places that are marked at the point of process execution when the event is missing (log move) or an event is not in the log but in the model (model move). This set of marked places (henceforth referred to as *marking*) is added to the alignment that is used as feature for the decision tree. One example for a feature of such shape is (('b', '»'), '[p1:1]'), which means that a model move for event b has been identified when the place named *p1* is marked. This is exactly the situation before the model repair in figure 4.3. The distinction between the alignment move including the marking and excluding the marking is important to be made in the context of a manufacturing process because it makes a difference at which place an operation is carried out. This is not so much the case for administrative processes that are used in the evaluation of the aforementioned examples in the work by Dees et al. There, every alignment that has the same type and event is repaired if it is predicted to have a positive impact on the process performance disregarding the location.



(a) Petri net with transition b as loop



(b) Petri net with transition b as optional transition

Figure 4.3: Different ways to insert a transition

These two extensions are referred to as `avoid-flower` and `move-loc`, respectively. The next section provides a more formal and detailed description of the repair algorithm as is used in the scope of this thesis.

4.3 Implementation

The concept has been implemented using the Python library PM4Py by Fraunhofer [77, 78]. The code of the implementation is available in a repository at https://github.com/simonthecreator/process-model-repair.

4.3.1 Requirements

The model repair procedure requires three parameters as input. Firstly, the model that is to be repaired, the second one is the alternative log that the model is compared to in order to get alignment moves and options to adapt the model. Henceforth, this log is referred to as *alternative* log because in the analysis, it might be necessary to discover the original model due to a lack of domain expertise and it is easier to distinguish the alternative log from the log for discovery of the original model. Lastly, a threshold must be defined that separates the traces into "satisfactory" and "unsatisfactory" traces based on the target KPI value.

Original Model N: The original model to be repaired can be created either based on domain knowledge or variants that cover a certain percentage of all cases. In the latter case the focus should be on including as many cases as possible with as few variants as possible, e.g. 40 % of cases are covered by only 4 variants with 753 variants in total in the event log. This is possible for the data from BPI Challenge 2020. An intermediate solution between a model drafted by a domain expert and a purely discovered model is when some knowledge can be used, e.g. to distinguish activities that must not be part of a planned model since they are never planned and hence always denote a special case such as production disturbance. The degree of generality or specificity has an impact on the further repair procedure. A general model as a starting point will result in a higher number of deviations which provides a larger data basis for the learning algorithm but it can also make the model under-fitting the training data because of noise.

Alternative Log L: The alternative log can be represented by process execution that was actually captured in production. If the original model was built based on the most common variants, the remaining log can serve as alternative log. The alternative log needs to include KPI values per trace which serves as the target value for each observation (process instance). We aim to create a repaired model that achieves improved KPI values compared to a not-repaired model also on log data that the model has not been repaired upon, i.e. it has not seen before. Hence, we split the alternative log into train and test data. Train log is applied to identify alignments between the log and the model. Test log is used to see if KPI values improve for the repaired model. The exact evaluation procedure is described section 4.3.3.

KPI Threshold k: We need to define a threshold value to separate the traces in training data into those that we want to consider for model repair and those we disregard. Naturally, this value can have a positive or negative meaning, e.g. good output quantity vs. throughput time so either traces with KPI values below or above this threshold are considered. Furthermore, it is possible to use a categorical KPI with discrete values. In this case, traces with a KPI value that is within the list of acceptable values, are considered good process instances. For the data that is analyzed in this thesis, only continuous, numeric values are used. This means that setting one threshold splits the data into two groups: acceptable traces and traces that are unwanted for model repair.

4.3.2 Procedure

The procedure described here is based on Dees et al. [1] and enriched with the extensions presented in section 4.2. This methodology seeks to adapt an original process model N to a new process model N' if the original model cannot replay an alternative log L entirely. One aim is to keep N' as close to the original model N as possible. The first step is to identify alignments between the (original) model and the alternative log. The alternative log L is split into a training

set L_{train} and a test set L_{test}. One trace represents one observation for the learning problem, hence alignments are identified per trace. This is depicted in Algorithm 1 which is executed with L_{train} as second parameter `alternative log`. For each trace in the log $\sigma \in L_{train}$, we identify which events need to be skipped in or added to N to be compliant to L_{train}. In particular, we identify the location at which N should have had a transition to replay each of the events in L_{train} using the Dijkstra Less Memory algorithm. Let $L_{train}A_N$ be the universe of all alignment moves for N and L_{train}. The alignment moves per traces are denoted as $\gamma \in L_{train}A_N$. As mentioned in section 3.1, there is a cost function to prefer certain alignment moves over others. By default, the costs for log moves and model moves are higher than for synchronous moves by a factor of 1000 (thousand) in GetAlignmentsAStar(). This is because we only want to add model moves and log moves to γ if it is necessary, i.e. if there is no way to express an alignment by only using synchronous moves. If GetAlignments() is executed with `includeLoc` set to *True*, then each move in γ does not only include the move transition but also the places that are marked in this point of model execution as described in section 4.2.

Algorithm 1 Get Alignments Per Trace

```
1:  procedure GetAlignments(original model N, alternative log L, boolean includeLoc)
2:      Γ ← ∅
3:      for each σ ∈ L do
4:          γ ← GetAlignmentsAStar(σ, N)
5:          if includeLoc then
6:              for each move ∈ γ do
7:                  AddLocationToMove(move, γ, N)
8:              end for
9:          end if
10:         append γ to Γ
11:     end for
12:     return Γ
13: end procedure
```

The alignments are then transformed into a one-hot-encoding with possible log moves and model moves (+ optionally location) as features. Results for using only moves and moves + locations are compared in section 5.2. From this one-hot-encoded observations, a decision tree is created. This tree is a regression tree if the target (KPI) value is continuous and a classification tree if the target value is discrete. The latter case does not occur in the scope of this thesis. At this point *repairing* refers to repairing the log and not the model. The adaption of the original model is made subsequently based on the repaired log.

The decision tree represents rules which are the basis for further classification of traces. How repaired logs are created from this decision tree is discussed in section 4.1. It is shown as pseudo-code in Algorithm 2. For each trace in Γ which contains the traces and the respective alignment moves, we traverse the tree to obtain the list of rules that define which moves are to be added in order to reach a leaf node with a desirable value. All traces that have the same list of rules (i.e. list of moves) are stored in the same cluster. The *ruleList* is the key referring to the list of traces (σ) + predicted values that belong to this *ruleList* in the container *logCluster*. For the execution of TraverseTreeToPredictedValue(), we make sure that the tree is traversed to the leaf node that represents the predicted value for this specific σ and no other leaf node that might contain the same predicted value. Hence the traversal of the tree results in a *ruleList* that is specific for σ.

Now all traces that actually have desirable KPI values and are predicted to result in desirable

Algorithm 2 Cluster Log By Decision Tree Rules

```
1: procedure ClusterLog(alignmentsPerTrace Γ, decisionTree tree)
2:     logCluster ← ∅
3:     for each σ ∈ Γ do
4:         predictedValue ← Predict(σ, tree)
5:         ruleList ← TraverseTreeToPredictedValue(predictedValue)
6:         logCluster[ruleList] ← (σ, predictedValue)
7:     end for
8:     return logCluster
9: end procedure
```

KPI values based on the alignments, can be used to repair the log. It can be seen in Algorithm 3 that log moves that are associated with desirable behavior are included into the repaired log while desirable model moves are not included to the log. This is because model moves are not in the log and if this absence of an event is predicted to have a positive impact, it should stay this way. Here the question occurs if the model repair procedure should be allowed to remove unwanted events from the original model or whether it shall only repair model moves by making the event optional. For the scope of this thesis, we stick with the latter solution as it is presented by Fahland and Van der Aalst [20]. Consequently, a model is never reduced, i.e. no transitions are removed because we assume that each activity in the original model has been placed with a purpose and must not be removed. Thus, model moves are repaired such that the event remains in the model but traces that do not include the event are replay-able as well. Synchronous moves are always added to the repaired log which means that events that are already in the same place in the log and the original model, are confirmed since the events are now more frequent in the entire log which makes them more robust when e.g. a frequency filter is applied.

Algorithm 3 Repair Log By Cluster

```
1: procedure RepairLog(logCluster, κ, k, Γ)
2:     log_repaired ← ∅
3:     for each (ruleList, cases) ∈ logCluster do
4:         if IsSatisfactory(κ(ruleList), k) then
5:             for each σ in cases do
6:                 σ_repaired ← ∅
7:                 for each move ∈ Γ[σ] do
8:                     if move is log move or synchronous move then
9:                         append move to σ_repaired
10:                    end if
11:                end for
12:                log_repaired[σ] ← σ_repaired
13:            end for
14:        end if
15:    end for
16:    return log_repaired
17: end procedure
```

The result of Algorithm 3, $log_{repaired}$, is at this point only a list of event lists. Complete log data is created from this list of lists containing generated timestamps in order to obtain a chronological order within a repaired trace. This is represented in Algorithm 4 as function

TransformToEventLog(*RepairedLog*) in line 6. The timestamps are not meant for (time) analysis but only to create a sequence of events that can be used for further operations. As mentioned in section 4.1, wrongly classified traces are disregarded which is here represented as function RemoveWrongClusteredCases(Γ, κ, k) in line 4 of Algorithm 4.

Algorithm 4 Get Repaired Log

```
1: procedure GetRepairedLog(alignmentsPerTrace Γ, KPIPerTrace κ, threshold k)
2:     tree ← CreateDecisionTree(Γ, κ)
3:     logCluster ← ClusterLog(Γ, tree)
4:     RemoveWrongClusteredCases(Γ, κ, k)
5:     RepairedLog ← RepairLog(logCluster, κ, k, Γ)
6:     TransformToEventLog(RepairedLog)
7:     return RepairedLog
8: end procedure
```

The next step is to integrate the repaired log into the (original) process model. This is represented in the lines 4 - 7 in Algorithm 7. There are different approaches to do this. The repair procedure by Dees et al. [1] relies on the repair technique by Fahland and Van der Aalst [20]. It is complemented by the choice structure repair by Qi et al. [45]. Furthermore, the decision whether we want to use the avoid-flower extension comes into play here. If we want to create a model that aims to not add transitions to a place like blossoms around a flower, then AddEachTransitionAsSkipable() is called which takes care that each transition added is accompanied by a silent transition that makes the added activity optional as well as a new place so that the input place of the new transition is not the output place as well. If we do not want to add new transitions in the avoid-flower way (i.e. if the boolean *avoidFlower* is false, then we rely on the RepairSubprocesses() procedure as it is provided by Fahland and Van der Aalst [20]. This decision is wrapped in the procedure named IntegrateSubprocessIntoModel() which is represented in Algorithm 5 and is applied in line 6 of Algorithm 7.

Algorithm 5 Repair Sub-Processes

```
1: procedure IntegrateSubprocessIntoModel(repaired event log L_repaired, model
   N_choice, boolean avoidFlower)
2:     if avoidFlower then
3:         N_repaired ← AddEachTransitionAsSkipable(L_repaired, N_choice)
4:     else
5:         N_repaired ← RepairSubprocesses(L_repaired, N_choice)
6:     end if
7:     return N_repaired
8: end procedure
```

The further main steps in Algorithm 7 are addLoops(), which tries to find repeated behavior that is to be included, RepairChoiceStructure(), which aims to repair log moves in choice structures more elegantly, IntegrateSubprocessIntoModel(), which groups behavior that is to be added at the same place in the existing model, and RemoveUnusedParts() which aims to keep the model close to real behavior. The subprocesses that are to be inserted need to be discovered from a respective subprocess-log. This is done using the ILP Miner. The cost function for finding alignments is adapted after the loop repair step was carried out. Costs for log moves are increased to make sure that a set of moves is found that aligns the log to the model without adding log moves, if there is such an alignment. The goal is that activities that are in a repaired loop structure are no longer identified as log move in the optimal alignment. The

removal of unused parts (step 3) is done by excluding parts that make sense according to the repair technique but are too infrequent to actually improve the model. This can occur especially when we test with log data that the repair decision tree has not seen before. The main steps of the technique by Fahland and Van der Aalst are discussed above in 3.2.

Alternatively to model repair, the repaired log can be used as input for "classic" discovery algorithms such as inductive miner or heuristics miner. Model repair needs a relatively small amount of changes in order to create the desired model compared to discovering a whole new model from *RepairedEventLog*. Different repaired or newly discovered models are compared regarding different quality criteria in the evaluation part of chapter 5.

4.3.3 Evaluation

As already indicated, the traces to evaluate the repaired model are not the same as the train data the repaired log has been created on. This is ensured by splitting the logs into training and testing data. The evaluation procedure is presented in Algorithm 6. We identify which traces from the testing logs L_{test} are compliant with the original model N and traces that are compliant with the repaired model N'. The latter should be more since the repaired model usually allows for more behavior except if many unused parts have been removed. In order to compare with other approaches, such as model discovery from the repaired log, traces that are compliant which each model that will be compared to the others, are identified. This is done the same way as depicted in line 3 in Algorithm 6 with replacing N by another model, e.g. a discovered Petri net model. We compare the average KPI value of all these sets of traces (line 4). This includes the mean, median, standard deviation. In addition to the performance of logs that are compliant to the adapted or newly created models, we are interested in the evaluation of model characteristics such as model replay fitness, precision, generalization and simplicity (line 5). As mentioned in section 2.3, this list is completed by a similarity measure. Similarity is included as model quality to support the idea that the repaired model is intended to be closer to the original one than a newly discovered model and similarity is a measure to express this in numbers.

Algorithm 6 Evaluation

1: **procedure** EVALUATION(L_{test}, N, N')
2: $\Sigma \leftarrow$ each $\sigma \in L_{test}$ if σ is compliant to N
3: $\Sigma' \leftarrow$ each $\sigma' \in L_{test}$ if σ' is compliant to N'
4: Evaluate KPI values of Σ and Σ'
5: Evaluate Model Quality Aspects of N
6: **end procedure**

Algorithm 7 provides an overview of the process repair and evaluation steps. After the train/test split, alignments are created and used to repair the model which is then evaluated as described in the paragraphs above.

Algorithm 7 Complete Procedure

1: $L_{train}, L_{test} \leftarrow L$
2: $\Gamma \leftarrow$ GETALIGNMENTS(N, L_{train})
3: $RepairedEventLog \leftarrow$ GETREPAIREDLOG(Γ, κ, k)
4: $N_{loops} \leftarrow$ ADDLOOPS($RepairedEventLog, N$)
5: $N_{choice} \leftarrow$ REPAIRCHOICESTRUCTURE($RepairedEventLog, N_{loops}$)
6: $N_{subprocess} \leftarrow$ INTEGRATESUBPROCESSINTOMODEL($RepairedEventLog, N_{choice}, avoidFlower$)
7: $N' \leftarrow$ REMOVEUNUSEDPARTS($RepairedEventLog, N_{subprocess}$)
8: EVALUATION(L_{test}, N, N')

5 Case Study

After the introduction to the model repair technique in the previous section, the technique is applied to process data in case studies. This section describes the data that is used to evaluate the techniques and its new extensions. Furthermore, it is discussed how the data is prepared to gain sensible insights and ensure that correct conclusions can be drawn. For the evaluation of the concept, "real world" process data from manufacturing organizations and publicly available data about administrative processes is used.

5.1 Data Exploration and Understanding

The manufacturing data is provided via a research project. In this project, a MES provider supplied process log data from manufacturing industry. The data is provided via a software for manufacturing controlling in an uniform shape. Moreover, the software allows to use KPIs like OEE as described in section 3.3.

We analyze two case studies for this thesis. **Case study 1** deals with an organization in manufacturing industry. The manufacturing data is complemented with administrative process data which is referred to as **Case study 2**.

The next section describes data cleansing and pre-processing steps that are carried out to ensure a valid and sensible result based on correct and complete data.

5.1.1 Data Preparation

The manufacturing data requires adaptions before it can be used as intended for this thesis. These are performed in three steps that are listed in the following bullet points. Note that the administrative process data is made available by the BPI Challenge in cleansed form (e.g. no incomplete cases) and does not require further preparation.

- **Remove Incomplete Cases**

  To attain sensible and correct results from analysis, the data structure must be consistent. Hence, it is required to remove orders (cases) that are incomplete, i.e. those that are still running at the time when the data has been extracted from the software system (snapshot). This is compliant with the work by Fahland and Van der Aalst where the removal of incomplete cases is explicitly mentioned as a preparation step [20].

- **Exclude Incorrect Timestamps**

  As mentioned above, sometimes feedback is reported afterwards automated to avoid "open" operations that are not completed in the data. We exclude the timestamps that have been reported this way because we only want to use manually created timestamps that reflect the actual work procedure. We assume that the timestamp of manually reported feedback is correct and that the automated timestamps are incorrect. One more reason to do that automated feedback has the same timestamp even though the corresponding process step is not parallel. Note that by excluding feedback with incorrect timestamps, operations that have only such incorrect feedback are excluded too. However, this is only the case if the actual timestamp is required, e.g. for the traces that are compared to a model to get alignments and subsequently repair the model based on these traces. For creating a reference model, planned timestamps are used which means that incorrect actual

timestamps do not harm the planned model. Hence operations that have only feedback with incorrect timestamps are still considered for discovery of a reference model.

- **Calculate OEE per Case (Order)**

  Attributes used for calculating OEE are duration and output. These attributes are tracked per operation which means that they need to be transformed in order to be an attribute of the order (trace). This is done by calculating the OEE as it is described in section 3.3.2 per operation and then using the mean of the OEE of all operations of an order as value for the order. Although it is not very precise to use mean values, it is still better than to use median (because we want to consider outliers, e.g. due to production disturbance) or sum (which would falsify the data as longer orders with more operations would have higher values).

- **Handle Outliers in KPI Values**

  One factor for calculation of OEE is performance. Performance is defined as the ratio of processing duration divided by planned processing duration, i.e. if these two durations are approximately of same length, then performance value will be around 1. However, the performance attribute contains outliers, i.e. values that are far higher than 1. The range of values is supposed to be between 0 and 1 or slightly above 1. The reason for outliers can be faulty or wrong tracking of processing duration or errors in planning. We remove these outliers by setting performance values to 1 for all process steps that have a performance attribute > 5. We do not want to remove outliers since each process step is part of a trace and for model repair we either incorporate a whole trace to the model or not at all. Therefore, each process step can be a valuable part of a trace even if performance of this very step is an outlier. Furthermore, for many materials, the number of orders (i.e. process instances) is low. We aim to not further reduce the number of processes that are suitable for model repair but want to retain as many processes as possible. If outliers would just be removed, even fewer processes of suitable size would remain. It is desirable to use as many traces as possible because the split into training and testing data, only a small fraction of all traces remains as testing traces at the end. It is easier to draw conclusions if the - possibly improved - KPI values are represented by many traces.

Before discussing results of the model repair case studies, the next section is about which processes are selected for analysis. This is not straight-forward as there is a large quantity of process data available and various criteria are considered to identify suitable processes.

5.1.2 Selection of Processes

In the large amount of process data available, the question occurs, which particular processes should be selected for model repair evaluation. We want to pick processes that are suitable to evaluate the repair technique presented in the previous chapters. It is particularly interesting to analyze those processes where model repair makes an impact regarding improved KPI values and this impact is not randomly but reflects reality. The goal is to improve the actual processes in "real life" by implementing adaptions suggested by model repair. To reach this goal, processes must be of a certain size regarding instances and process steps. The size requirement is essential to select processes that make sense to be analyzed from a process mining point of view. It would not be sensible to repair a process that consists of only two or three distinct process steps or only has a few different instances. Furthermore, in order to check if traces that fit repaired model A have better KPI values than repaired model B or the original model, a sufficient number of test traces is necessary which means that a large number of traces per process is desirable.

Additionally, the idea occurred to carry out association measures to detect a possible correlation between process instance behavior and an instance attribute, as presented by Leemans et al. [79]. The attribute is in this case the value of the KPI. The association measure has been computed by executing the plug-in "Compute association/correlation between the process and trace attributes" implemented by Leemans et al. in the ProM framework [80][1]. However, the idea has been abandoned since processes with a high association measure do not lead to better results regarding improved KPIs than those processes with poor association measure scores. It turned out that for the selected processes the scores of correlation between process behavior and OEE as well as all factors of OEE are low. One reason might be that the association measure is based on randomly chosen pair of traces to represent the trace behavior and hence the Process Versus Numerical Attribute Association as it is available in the ProM framework is not applicable on process data with rather few process instances and high variability such as the manufacturing process data used in this thesis.

As mentioned in the beginning of chapter 5, the manufacturing process data is supplied by a MES provider. The data set comes from four different production sites. Each site produces other goods and has different process characteristics. On a comparison between the four sites, not only the number of orders per material ID differ strongly but also the share of orders that a single material ID covers. For instance, in site 2, the three largest material IDs account for far more orders than the other materials in this site. The figures are similar for site 3 where few material IDs cover around 1000 orders each. The data is different for sites 1 and 4: they both have only small processes with largest material (ordered by number of orders) of site 1 consisting of 64 orders and 13 distinct operations and 92 orders and 4 distinct operations for the largest material of site 4. Moreover, site 4 has no process that has more than 6 distinct operations among those that have at least 10 orders. Due to this data structure, processes of site 1 and 4 are not considered for evaluation. The two cases presented in case study 1 in section 5.2 are both part of site 3 because these cases are best suited to discuss results. However, other material IDs from site 2 are used for model repair and we refer to these results when the impact of the extensions `avoid-flower` and `move-loc` is discussed.

An administrative process complements the evaluation data to see possible differences to manufacturing processes. As the main focus is not on improvement of administrative procedures, the administrative data is limited to one process. Additionally, more domain knowledge of the administrative process would be required for more thorough analysis of the respective data. In table 5.1, the selected processes on the basis of which the evaluation was made, are displayed. The table shows the number of process instances (i.e. "orders" for manufacturing data) and unique events ("operations") as well as the number of distinct variants that occur in this process. A variant represents the exact way the process is executed (i.e. sequence of events). That means if there are multiple instances that represent the same sequence of events, this is one variant.

Table 5.1: Selected processes for case study

Case Study	Process	# Instances	# Events	# Variants
1	Manufacturing - case A	1183	8	696
1	Manufacturing - case B	510	12	250
2	Request For Payment	6886	19	89

[1] ProM Tools are available at https://promtools.org/

5.1.3 Role of the Original Model

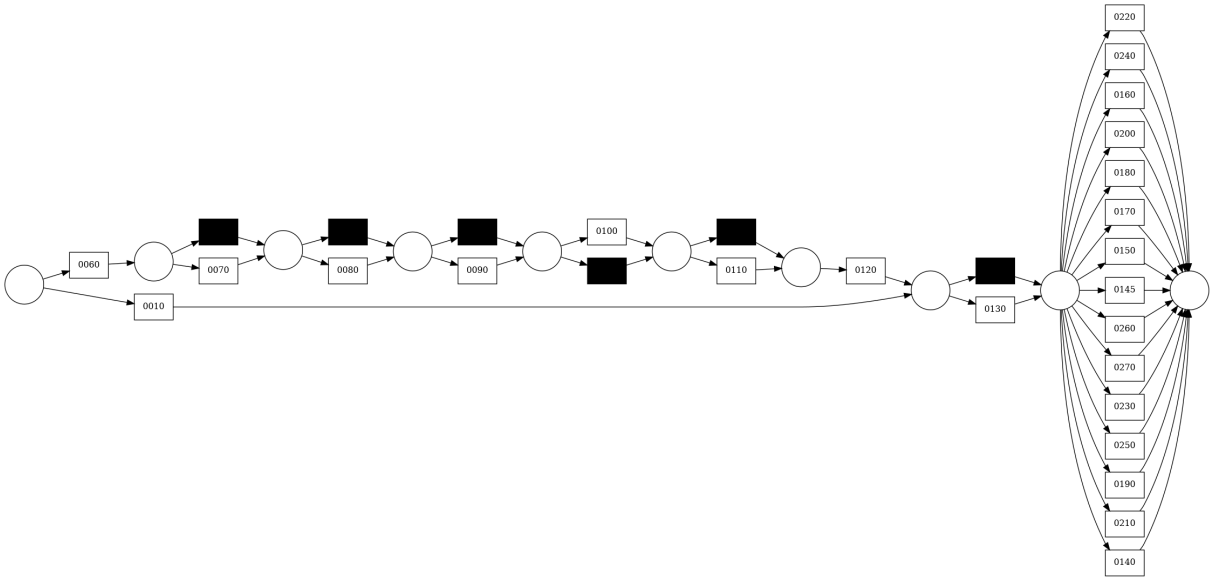
As already mentioned, the original model as starting point for process repair has a large influence on the outcome. A rather complex original model, that incorporates many variants, is likely to require fewer adaptions during model repair than a simple original model. Since a complex model already represents diverse behavior, there will probably be only few deviations between the model and the training log. This makes it harder to compare the impact of model repair with test data because only few test traces will fit the repaired model that do not already fit the original model too. In contrast, if the original model is very simple and straight-forward, many adaptions will be made during model repair which can again lead to only few test cases that fit. In other words, the original model influences the alignments which are the basis for model repair as alignments represent (possible) repair steps. The original model and how it is created are important parameters to model repair alongside costs for repair steps during alignment analysis and possible threshold to filter infrequent repair steps.

So far, two alternatives to create the original model have been used. The first is to create a general model that represents the planned process from a business perspective. It includes all activities in the order they are planned to be executed which is the operation number in ascending order. Each activity occurs only once. Rework operations are excluded beforehand. It is a reduced original model that represents a sequence of the most common activities rather than a combination of the most common variants. The second option is to discover a model from the prepared log (without rework operations) using inductive miner. Throughout this thesis, parameter noise threshold is set to 0.8 when the inductive miner is used to discover the original model. This approach comes close to using the most common variants. Furthermore, it enables to measure similarity because the inductive miner guarantees soundness which allows to transform the model to a process tree. The process tree in turn allows to measure the edit distance to other process trees.

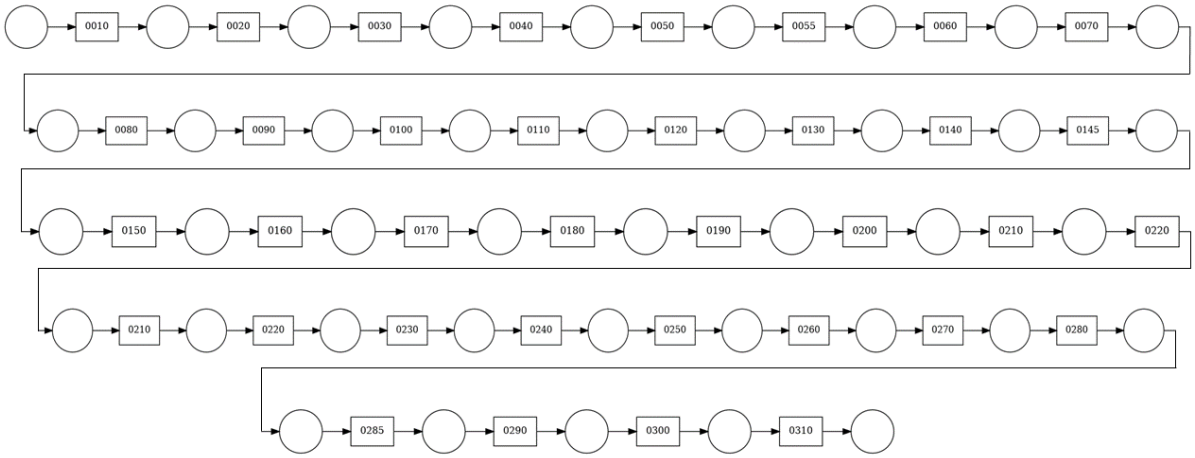
Let us now look into an example with many different, very short traces to be used for creating the original model These very short instances often consist of only one event, which leads to unorthodox models when we want to represent the most common behavior. For instance, in case C in the manufacturing data, 8 variants cover 30 % of cases but all of these variants are just one event and this event is of course different for each variant. At first glance, this might seem like a reason to use the "planned" process model as described above. However, in contrast to the processes discussed in case study 1, it is not sensible to use the planned original model here. The example of case C is illustrated in figure 5.1 and shows why: The discovered original model for this process is "spaghetti-like". The very short instances with diverse events lead to constructs like on the right of figure 5.1a. Hence a model like this is inapplicable as original model because even though there are many transitions, the model does not reflect much behavior as most transitions are optional or part of an exclusive choice on the very right part of the model. However, the planned original model depicted in 5.1b is not better suited for model repair as it is a long chain of transitions and probably no trace in the test log fits the model even after transitions have been added during repair. The variety of variants makes it difficult to use processes like this for the model repair procedure presented in this thesis.

However, many times it is still a better choice to use a discovered model than a planned model because a chain of transitions as depicted in 5.1b is even less likely to fit test traces. Therefore, in all cases that are discussed in section 5.2, we focus on the analysis of model repair based on the discovered original model. Note that both the planned and the discovered original model for case C are "extreme" examples and the discovered original models that we use in sections 5.2 and 5.3 are simpler. Furthermore, if there is more domain knowledge available that can be used to draft a original model as reference, it will be advisable to leverage this because then the model repair is more stuck to reality and process stakeholders are already involved.

There is a third variant that uses the most common variants among the traces. The goal is to cover as many traces with as few variants as possible so that e.g. 4 out of 50 variants cover 40 % of all traces. Naturally, this coverage of variants is only applicable when there are common variants which is not the case for the manufacturing processes that we looked into so far. However, this is a suitable way to create a reference model to start with for administrative processes which we show in section 5.3.



(a) Original model discovered with inductive miner



(b) Planned sequence of activities using domain knowledge

Figure 5.1: Manufacturing Data - case C: original models created in different ways

Now that an overview about data content and preparation has been given and the selection of processes as well as the role of the original model have been discussed, the remainder of this chapter presents the application of model repair and evaluates the results. Aspects of applicability and quality of model repair techniques are discussed. The following sections compare different types of processes with different repair settings. The validation is guided by criteria that assesses the fitness of the model to the log and average KPI values of traces that can be replayed by a model version. Furthermore, we consider model criteria that describes characteristics of the sole model without reference to log data, such as simplicity and similarity to the original model.

Throughout the discussion of the case studies, we use tables to give an overview about the model quality characteristics. In these tables, the models are referred to with abbreviations/names that are as follows: the `original` model is the unedited model that is the starting point of model repair. The `kpi-based` model is the output of the repair procedure that follows the example by Dees et al. [1] as it uses the mere occurrence of a (log/model) move for an event as feature to learn a decision tree that splits traces into good and bad with the KPI as target value. The outcome is compared to the model that uses all traces (i.e. repairing all deviations between model and log) independently of any KPI value during model repair, which is referred to as `repair-all-traces`.

The models that are output of the **extensions** to model repair (introduced in section 4.2) are referred to as `move-loc` and `avoid-flower`. The move-loc procedure, the features for the decision tree that decides which traces will be considered and which not, include not only the move (i.e. log or model move for a certain activity) but also the location of this move. In contrast, the procedure that follows the example by Dees et al. [1] only uses the move itself as feature. This means that only the presence of a move for a certain event is used as feature no matter where in the model this move appears.

The avoid-flower extension uses the same repair steps as are used to obtain the `kpi-based` model (following the example by Dees et al. [1]) but aims to create a sound model that does not allow for infinite loops. In other words, it aims to avoid flower models which allow transitions to fire multiple times or even anytime. Lastly, we also apply existing discovery algorithms on the repaired log, namely the heuristics miner and inductive miner algorithms. The parameter settings for the heuristics miner is: dependency-threshold: 0.5, and-threshold 0.65 and loop-threshold 0.5. For the inductive miner, the noise-threshold is set to 0.0. Both algorithms are applied using the PM4Py Python library [77, 78]. The resulting models are referred to as `repaired-HEU` and `repaired-IM` in the evaluation tables.

As mentioned above, the evaluation of the model repair results are split into two case studies, namely manufacturing (case study 1) and administrative process (case study 2). Within manufacturing data, each case (i.e. process) is presented and discussed separately.

5.2 Case Study 1: Manufacturing

The manufacturing process data requires some clarification regarding the terms and expressions used. It is necessary to describe these terms as they are the building blocks of the process structure.

5.2.1 Glossary for Manufacturing Process

The following terms are important for the data structure of case study 1, i.e. manufacturing process data:

Material represents an item or product that is the output of a defined manufacturing process. It is not necessarily a finished product that is ready to be sold but can be an intermediate product as well that needs further processing. A material has a name and an ID. The latter is a unique identifier while the name is not necessarily unique. It is left to the organization that uses this data structure whether different versions of a product (e.g. different color) are represented as different materials in the data or if there must be a clearer distinction. For our analysis, a material corresponds to a **process** which means that only process instances that produce the same material are treated as comparable instances that are the input for one process model.

Order is an instance that describes the process that the material undergoes. Each order refers to exactly one material. In process log terminology an order is a **trace** (i.e. an instance of the process). An order has a planned sequence of process steps which means that it is possible to use planned process steps to create a reference model. This is an important feature as a reference model is a required input for process repair. Of course each operation has a timestamp of the actual execution so that a process can also be analyzed based on the real behavior.

Operation represents one step in a process, e.g. welding, machining, etc. In the event log terminology, each operation is an **event**. An operation can be split into set-up and processing phases. Additionally, there can be disturbance times for an operation. These different durations are important to calculate KPIs such as the OEE (see section 3.3.2). The different phases of an operation are marked by feedback of operators (workers). Each feedback updates the status of an operation. For example there is a feedback for the begin of set-up, for the end of set-up, begin of processing and so on. Although the feedback layer is even more detailed, we use operations as events since this allows to overcome data consistency problems. Operations can hold further information such as resource (worker, machine, workplace).

Feedback is the most detailed layer of manufacturing process data. It provides information about the status of an operation. The first and last feedback of an operation marks the begin and end timestamp of the operation. Although the analysis of feedback would allow for more detailed process analysis, the amount of events would become overwhelming and data quality issues would occur. These problems have been avoided by considering an operation as an event and use selected feedback timestamps as operation timestamps.

To sum up, a material is a process that is executed as orders which are process instances. Orders consist of operations that are the process steps and are represented as events in process logs and models. The reasons why this structure has been determined, are discussed in the next paragraph.

Orders are seen as process instance because there is a sufficient number of orders per material available and orders can be compared rather than operations. Operations can vary across different orders. This leads to incorrectly discovered processes if operations across orders (but still within the same material ID) are compared. Hence, operations are not treated as process instances, even though they would provide steps (begin set-up, end set-up, etc.). Typically, the variation between operations is stronger than for orders within the same material ID. One example can be machining operation which may be differently carried out dependent on the type of work item to be machined and hence dependent on the order. Furthermore, if an operation would be a process instance, a feedback would be an event. Since feedback is not mandatory to be created exactly at the time when the corresponding action was carried out, they are sometimes created afterwards. Hence, feedback as events does not guarantee to reflect the actual operation procedure. Feedback created afterwards may also have been generated automatically which may cause problems regarding timestamps as multiple feedback can be created with exactly the same timestamp. In these cases, parallel tasks might be incorrectly identified in process mining analyses.

Further aspects of manufacturing data, such as the relationship between materials and a resulting order sequence, or transports remain disregarded in the scope of this thesis. This is due to the quantity of data and the lack of domain knowledge that would be required to analyze these further aspects of the data in a sensible and correct way.

After clarifying the structure of the process data, the next section presents the first example of model repair. As mentioned in the previous chapters, we use OEE as target KPI for manu-

facturing processes in case study 1. The threshold to distinguish desirable traces from unwanted ones is set to the 25 % quantil of the OEE per trace over all traces for the process. This means that 75 % of all traces have a KPI that is greater than this threshold and are considered to have a "good" OEE. The OEE per trace is calculated before as the mean OEE of all process steps that belong to this trace.

5.2.2 Case A

Case Description

We start with a comprehensive example where we inspect each different model that is the output of the multiple ways of model repair. It is one of the larger processes of the manufacturing process data with 1183 cases. However, the process contains only 8 distinct activities which indicates that the (discovered) process models are not too complex. The train/test split is 67 %/33 % which means that there are 391 traces in the test data. The original model, that serves as starting point for model repair, is illustrated in subfigure 5.2a. The inductive miner algorithm is applied to discover the original model that balances simplicity and fitness to the log. The original model is very simple as there is only one possible path through the model. This means that there is a large "main street" process which covers most of the variants.

Results

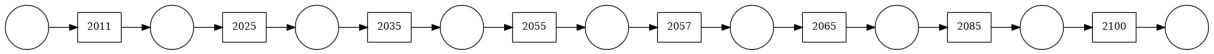
Graphic Models

The repair technique by Fahland and Van der Aalst [20] tends to create "clustered" transitions that share the same place as entry and exit point of the added behavior. This can be seen in the models: the KPI-based repaired model that follows the example by Dees et al. [1] in subfigure 5.2b; even more in the model that repairs all traces no matter if they are connected with a good or bad KPI value (subfigure 5.2c) and the model that uses the move and location as feature (subfigure 5.2d). These three models are rather complex with many added transitions which are highlighted in blue. Since the process has many variants, this indicates that the most common variants do not comply with the cases that have desirable OEE. At least the KPI-based repaired model (subfigure 5.2b) and the move and location model (subfigure 5.2d) would be simpler if the cases that are the common variants would also be those with good OEE. This is particularly clearly visible in this example because the original model is a straight-forward chain of events and the repair techniques add a lot of new behavior to this model in order to integrate desirable behavior.

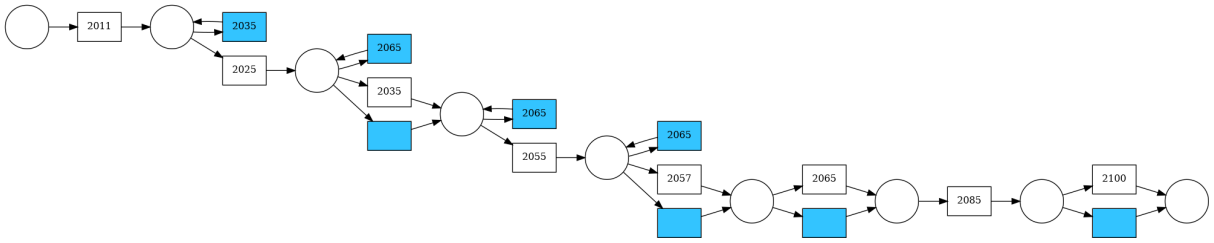
On the contrary, the adapted repair technique that is meant to avoid such flower models leads to a simple model that adds most transitions one after another. The result of the avoid-flower technique is a lengthened model with transitions in a sequence, each with a silent transition to make the actual transition optional (subfigure 5.2e). Note that subfigure 5.2e is rotated by 90 degrees which means that the process flow goes from bottom to top. This version integrates additional behavior into the model without creating transitions that have the same preceding and subsequent place but also has disadvantages such as transitions that are added in a row and can only be executed in this order which reduces the variety of possible behavior. Despite this restriction, the `avoid-flower` model leads to a reasonable result for this process as it fits the same 166 test traces as model `kpi-based`. This is discussed in more detail in the evaluation part of this process.

Lastly, the results of heuristics miner and inductive miner algorithms are also added for this process. They are displayed in subfigures 5.2f and 5.2g respectively and the process flow goes from bottom to top (like in subfigure 5.2e). There are no transitions marked in blue in these two models because they are not created by adapting the original model but are newly discovered based on the repaired log. The repaired log is the one that is used to repair the original model,

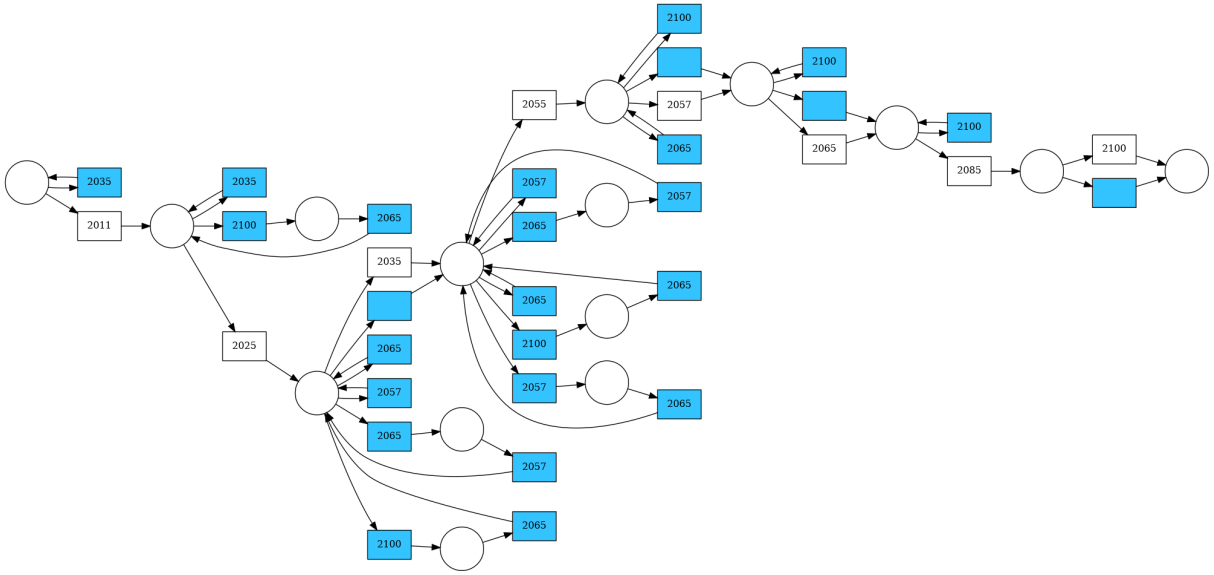
i.e. to make the original model compliant with the repaired log. The heuristics miner aims to incorporate the most frequent behavior while keeping the model simple. This leads to a rather simple process model but is not well suited for this example since only few test traces fit the heuristics miner model. The inductive miner is designed to guarantee soundness which typically leads to a more complex Petri net model with a lot of silent transitions. For case A, the inductive miner model is the only one that keeps up with `avoid-flower` and `kpi-based` when it comes to fitness of test traces. One possible conclusion from this is that the inductive miner is better suited to identify the most common behavior in the log even though the resulting process model appears to be inflated.



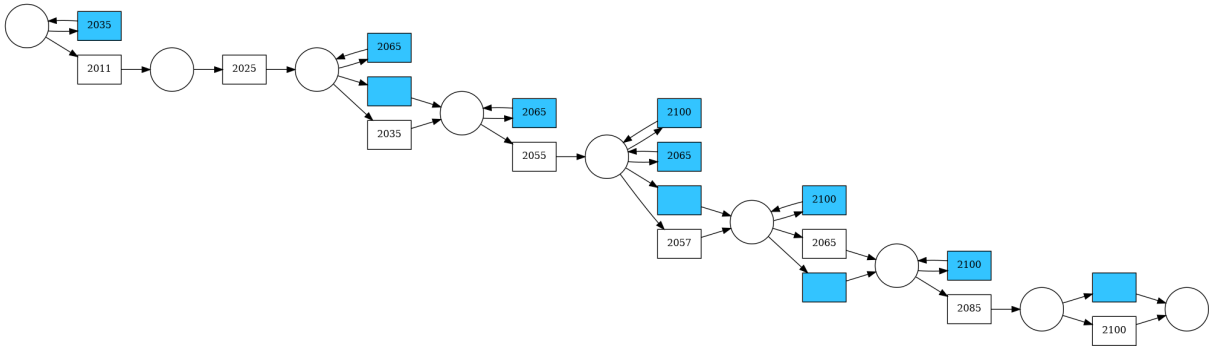
(a) Original model as starting point



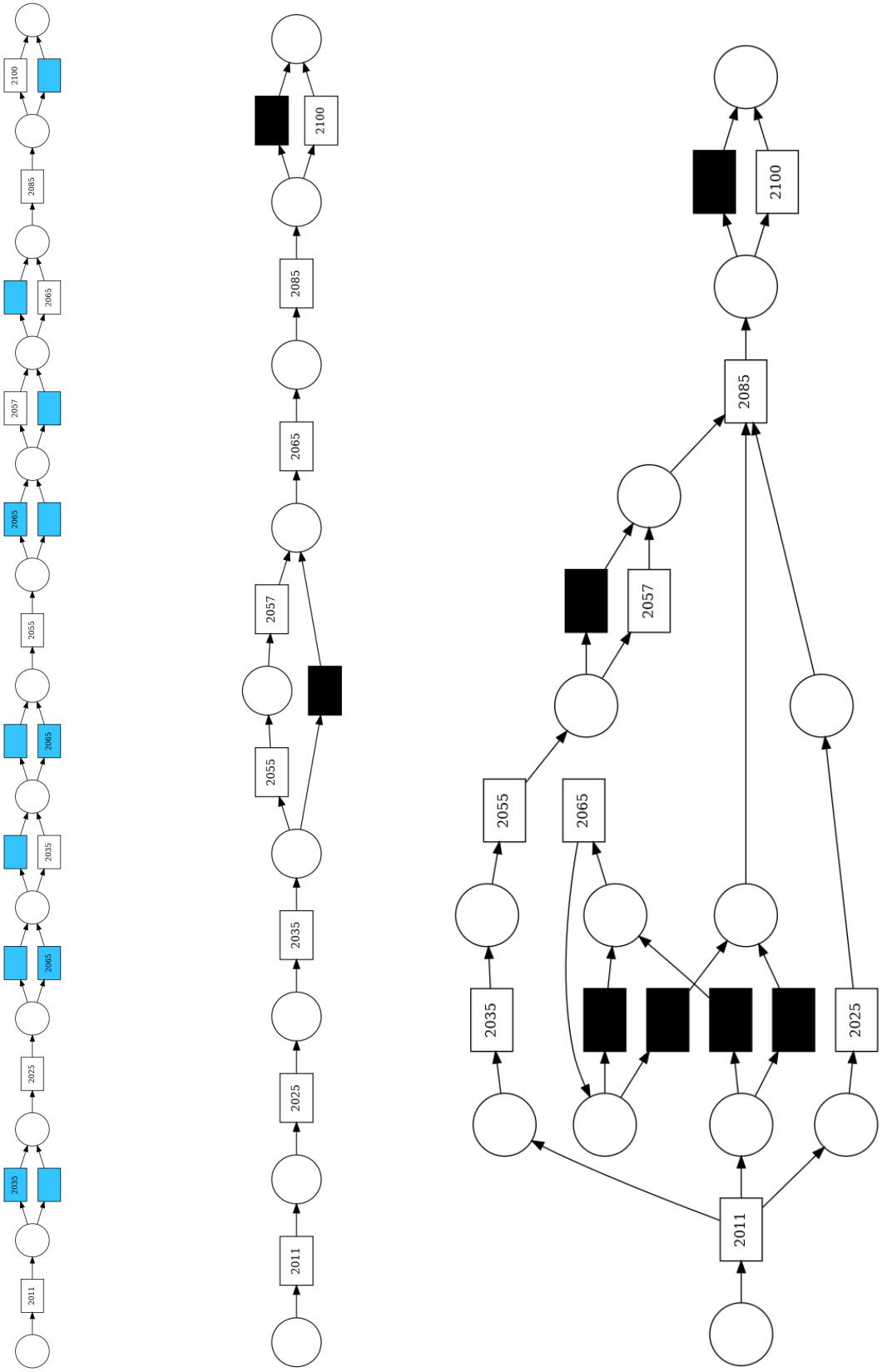
(b) KPI-based repair following the example by Dees et al.: move as feature [1]



(c) repair-all-traces using Fahland and Van der Aalst technique [20]



(d) move-loc: location and move as feature



(e) avoid-flower: model integrates same steps as 5.2b but avoids imprecise flower model

(f) repaired-HEU: model discovered by heuristics miner on repaired log

(g) repaired-IM: model discovered by inductive miner on repaired log

Figure 5.2: Comparison between Petri net models for case A

Table 5.2: Characteristics of different models for case A

Model	# Fitting Traces	Mean OEE	Median OEE	Standard deviation of OEE	# Repair Steps	Simplicity	Precision	Generalization	Log Fitness
original	114	1.3637	1.3349	0.268	0	1.000	1.000	0.949	0.887
kpi-based	166	1.3845	1.3780	0.255	8	0.641	0.843	0.631	0.923
repair-all-traces	391	1.3820	1.3704	0.268	20	0.563	0.545	0.478	0.992
move-loc	283	1.3819	1.3699	0.265	11	0.583	0.586	0.717	0.971
avoid-flower	166	1.3845	1.3780	0.255	8	0.702	0.848	0.702	0.946
repaired-HEU	114	1.3637	1.3349	0.268	—	0.905	0.993	0.843	0.888
repaired-IM	166	1.3845	1.3780	0.255	—	0.730	0.822	0.739	0.945

OEE Scores

Before we discuss the results that are listed in table 5.2, note that one main goal of KPI-based model repair is to choose repair steps so that the traces that fit the repaired model show better KPI values than if all possible steps would have been repaired. In the wording of this thesis, this means that the results for any of the repaired models `kpi-based`, `move-loc` and `avoid-flower` should be better than the results for model `repair-all-traces`. Case A partly meets this criterion. Note that this goal has not been achieved in the KPI-based process repair applied by Dees et al. [1]. In the work by Dees et al. the model that repairs all traces shows better KPI values. Table 5.2 shows that in case A, the test traces that fit the `kpi-based`, `avoid-flower` and `repaired-IM` model have a better mean and median OEE than model `repair-all-traces`. Furthermore, the mean and median OEE of `move-loc` is virtually at the same level as for `repair-all-traces` since the difference are only visible at the fourth decimal. A better mean and median OEE for either `kpi-based` or `move-loc` than for `repair-all-traces` is observed in 11 of 19 more cases that have been analyzed. Note that only selected cases are part of this thesis as there are challenges in data preparation and case selection as discussed in section 5.1.2, e.g. appropriate number of process instances.

The mean and median OEE for `kpi-based`, `avoid-flower` and `repaired-IM` models are marked in green since these are the best values for this process. The second column from the left shows the number of testing traces that fit the model. The model `move-loc` fits far more test traces than the other repaired models as well as `repaired-HEU` and `repaired-IM`. This suggests that the changes made during `move-loc` repair achieved to construct a model that matches a big share of process behavior ("main street" of a process). Apparently, some changes that were made in `move-loc` have a large impact on fitness of test traces since the `move-loc` incorporates only 3 repair steps more than model `kpi-based` but fits significantly fewer test traces. The only slight difference between `move-loc` (figure 5.2d) and `kpi-based` (figure 5.2b) can also be seen in the respective graphic representations. It is not surprising that `repair-all-traces` fits all 391 test traces as otherwise we could conclude a big difference between training and test traces which is unlikely.

The mean, median and standard deviation of OEE only differ slightly among all models. The median OEE is especially interesting to look at for the `kpi-based` and `avoid-flower` model. Here the difference to the `original` model is larger than for the mean OEE. This means that the repair steps applied for `kpi-based` and `avoid-flower` allow to include a number of traces into the model so that it fits the desirable test traces and achieves better average OEE. The

only model that fits more test traces than `kpi-based` is `move-loc` with slightly worse mean and median OEE. In 6 more cases, the fitness to the test traces is better for `move-loc` than for `kpi-based`.

The number of repair steps is used to measure how close a repaired model is to the original model. The fewer steps are performed, the more similar the model is to the original. Clearly, the integration of all traces triggers most repair steps (20). However, it is interesting to see that the change of the feature in the decision tree (implemented for model `move-loc`) triggers more repair steps than using the move only as feature. The model `move-loc` is reached by using more repair steps than for `kpi-based` in another 6 out of 19 inspected cases, in 10 cases the number of repair steps is equal and in 3 cases `kpi-based` triggers more repair steps. The number of repair steps is equal for `kpi-based` and `avoid-flower` since they aim to incorporate the same behavior just in different ways.

Model Quality Dimensions

It can be seen that the `original` model is the simplest, precisest and most general. This is plausible if one recalls the original model displayed in figure 5.2a. Simplicity is measured as the number of active transitions before and after the execution of a transition. The original model is extremely simple and hence more parts of it are used than for the other models (i.e. it is more precise). Except for the original model, high simplicity score does not necessarily correspond with a lot of fitting traces as `repaired-HEU` is a rather simple model but does not fit the test traces well. However, the models `avoid-flower`, `kpi-based` and the discovered model `repaired-IM` are pretty simple and have a acceptable fitness. This supports the idea that there is a lot of "main street" behavior in the process which is represented well by these models. The lower simplicity score for `move-loc` model is due to the larger number of transitions that are active when a place with many ingoing and outgoing arcs is active.

The effects of different discovery algorithms on model quality dimensions is discussed in literature [67]. Most of the models discussed here are not discovered but repaired. Nevertheless, the interpretation of the quality dimension is also applicable here as it is mainly about the relation between model and log. The precision score for `move-loc` is rather bad relative to the other models. On the contrary, `move-loc` has a good generalization score. The `move-loc` model allows for more behavior than the log requires (hence low precision) and but has only few infrequently visited parts (high generalization).

The model `kpi-based` has good precision and generalization scores. Precision and Generalization of `avoid-flower` and `repaired-HEU` are rather high due to the simple model structure. 12 out of 19 further cases also show a better precision for `avoid-flower` compared to `kpi-based`. One could conclude that the repair technique by Fahland and Van der Aalst - which is applied for models `kpi-based`, `repair-all-traces` and `move-loc` - is more focused on human readability than on model quality dimensions because the clustered transitions keep a model more lightweight than adding transitions as a sequence. This confirms that extensions to Fahland and Van der Aalst model repair technique are a practicable way to investigate the different aspects of model repair on process data from diverse sources. The intended outcome are adapted model repair techniques that lead to improved scores for certain criteria, such as simplicity, precision and fitness as this example shows. Of course this does not succeed in all cases but the intended better log fitness in `move-loc` as well as improved simplicity and generalization of `avoid-flower` compared to `kpi-based` is achieved in this example.

Intuitively, the fitness scores are expected to be higher for models that fit many test traces. This is only partly correct. The models that allow for a lot of behavior, i.e. `repair-all-traces`, `move-loc` and `repaired-IM` have the highest log fitness. This also reflects the number of fitting traces. However, log fitness for `repaired-IM` is better than for `kpi-based` even though they

fit the same number of test traces. This shows that model repair performs worse than model discovery (with the inductive miner) when it comes to log fitness as the discovered model is more shaped to the training log while repair adapts an existing model and is limited to existing behavior which is not overruled. The log fitness of `avoid-flower` is better than for the other models that fit 166 test traces (`repaired-IM` and `kpi-based`). One conclusion can be that the `avoid-flower` models fits more traces almost, but none of them completely.

The manufacturing process of case A appears to be as diverse as an administrative process even though it contains only 8 distinct events. The scores of the different repaired models can be interpreted in a way that certain behavior is required in different places and newly added transitions need to fire in a certain sequence. This can be seen by the relatively low number of repair steps necessary to create the models `kpi-based` or `move-loc`. This is intuitive for a manufacturing process, since it would not make sense to perform welding before machining as opposed to administrative tasks, e.g. writing an e-mail can rather be performed before or after a document has been forwarded. On the other hand, it might be desirable for manufacturing processes to keep a certain flexibility regarding the sequence of tasks/events. It might be only certain tasks that cannot be executed in flexible order from a technical point of view (e.g. welding/machining) while this might be possible for others. If these flexible tasks are the majority, this could explain the effect of clustered events that can execute virtually at any time in the process as can be observed in the repaired models, especially `repair-all`

Wrap-Up

In case A, the repaired models `kpi-based` and `avoid-flower` as well as `repaired-IM` have better average OEE scores than if all traces are repaired. This means that it is possible to repair the original model based on selected traces with high OEE scores in order to improve the average OEE score even more than if all traces are repaired. Again, this is emphasized because it has not been achieved in the work by Dees et al. [1]. The fact that the `move-loc` extension leads to a good result regarding the quantity of fitting test traces as well as OEE values shows that this extension is a sensible way to perform model repair for a manufacturing process based on the data examined in this thesis. This is also true for 5 other material IDs inspected, which show a better fit to test logs for the `move-loc` model than for the `kpi-based` model. There are also some difficulties in case A: The large share of behavior that is covered by only few variants ("main street" of the process) makes it difficult to adapt the model in a way that it remains easily readable for the human eye and at the same time incorporates sufficiently new behavior to achieve improved OEE values.

Moreover, the model discovered by the inductive miner algorithm leads to a decent result in case A. One conclusion from this can be that a discovery algorithm can be used when only few knowledge is available and when we want to leverage the advantages that inductive miner offers. Furthermore, among the range of process discovery algorithms available, the process owners can choose one that fits the needs of the intended repaired model. Process discovery can be applied on repaired log to achieve an improved model that also takes advantage of model discovery, such as explicit parameters of how complex the result should be.

5.2.3 Case B

Case Description

The next case is a smaller one consisting of 510 traces. The number of test traces that we use for evaluation is one third, i.e. 169. The number of distinct activities is 12 and higher than in the previous example of case A where it is 8. The number of variants compared to the number of traces is slightly higher but approximately at the same level as case A. Nevertheless, the results are interesting to discuss because we can inspect which differences in two models lead do

different results regarding test traces. The original model was again created using the discovery with inductive miner. In this case, the discovered original model differs from the planned original model. However, as already mentioned in section 5.1.3, we stick with the discovered model as it fits more test cases which is even more important when the number of test cases is low anyways.

Results

Figure 5.3 shows the different results as Petri net models for case B. As already mentioned, the original model, which is depicted in subfigure 5.3a, has been created based on domain knowledge and using the inductive miner. The next subfigure 5.3b shows the model that incorporates all possible repair steps.

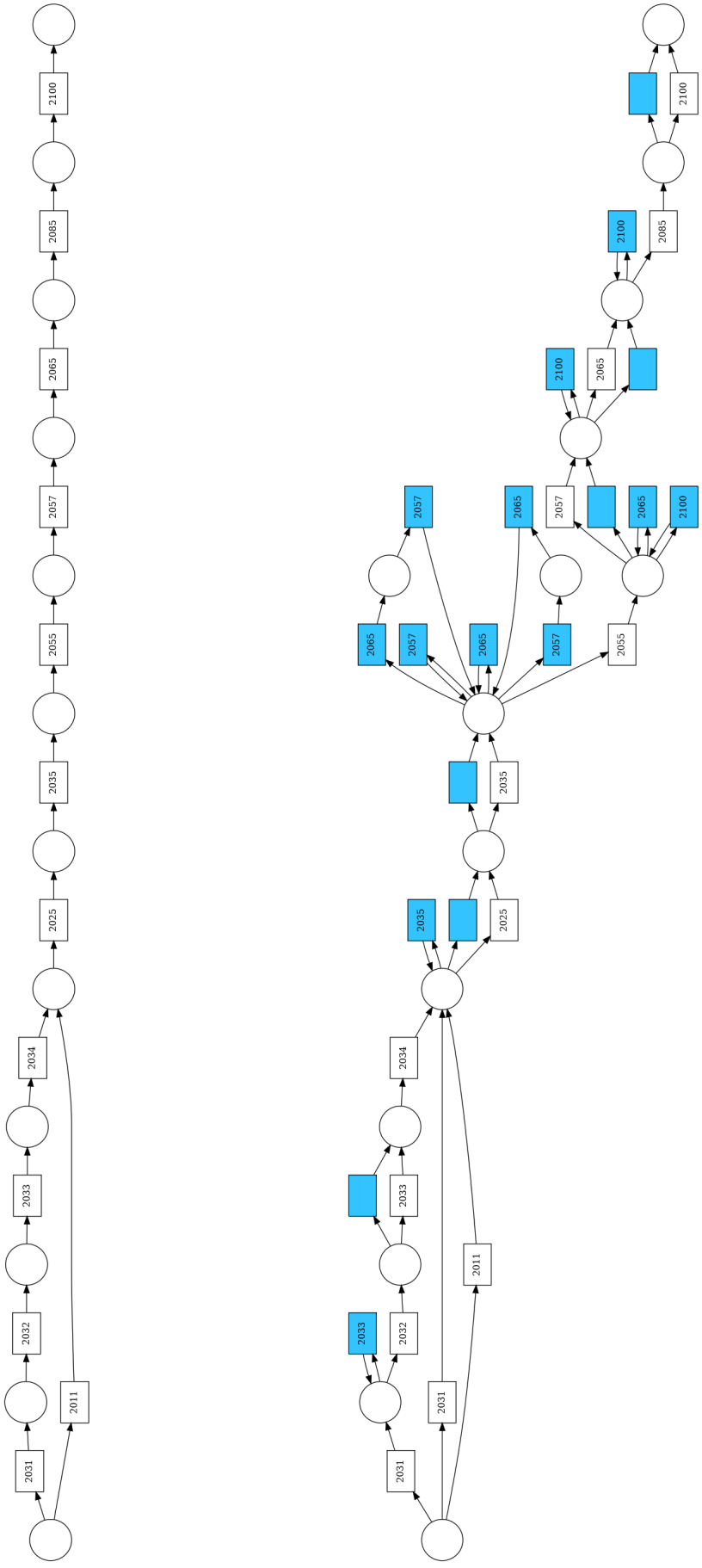
It is especially interesting to examine `kpi-based` (subfigure 5.3c) and `move-loc` (subfigure 5.3d). At first glance the models look similar with only few differences. The most obvious differences are the additional transitions at place $p7$. However, the figures in table 5.3 show that the `move-loc` model fits 20 traces fewer than the "regular" `kpi-based` model. There are 169 test traces in total, hence this is not a small difference. This difference is the reason why the mean and median OEE are lower for the `kpi-based` model than for `move-loc`. This shows that adding the location to the feature has a positive impact regarding average OEE - at least for certain cases. The mean and median OEE of `move-loc` is marked in light green as these are the highest OEE values among the repaired models (which excludes `repaired-IM` and `repaired-HEU`). The discovered models show the best mean and median OEE values with the mean of `repaired-IM` and the median of `repaired-HEU` marked in green as they are the highest values of all models listed in table 5.3. A better fitness to test traces for `move-loc` than `kpi-based` is observed in 5 more cases that have been analyzed. Like in case A, the `avoid-flower` (subfigure 5.3e) fits the exactly same test traces as `kpi-based`. The model `repaired-HEU` (subfigure 5.3f) performs poorly regarding the fit for testing traces.

It is in this example, the `repaired-IM` model (subfigure 5.3g) which fits most test traces - if we disregard `repair-all-traces` (subfigure 5.3b). However, the difference of fitting traces between `repaired-IM` and `repair-all-traces` is only small. These findings indicate that only few traces have very similar or even the same behavior. The performance regarding average OEE of the discovered models is relatively good. Like in case A, the heuristics miner does not perform very well regarding trace fitness but has the best median OEE. The inductive miner is much better in preparing a model that fits the test data. Again, the reason for the good fit of the inductive miner model might be that the IM is more focused on maintaining a sound model that fits more cases even if they are diverse because the IM keeps on adding behavior until the defined noise threshold is reached.

The results compared in figure 5.3 show that in case B, `move-loc` leads to more adaptions than the version that uses the sole move as feature (`kpi-based`). This is also true for the previous case A. One reason might be that due to the more precise division into satisfying and dissatisfying traces, the set of adaption steps is larger when one trace is repaired. The decision tree develops more nodes and hence more moves are considered. Consequently, the `move-loc` models are not as simple, precise and general as the `kpi-based` models as can be seen in table 5.3. Clearly, a Petri net model with more transitions is more complex and allows for more behavior which diminishes precision. Like precision, generalization is also reduced since more parts of the model are infrequently visited.

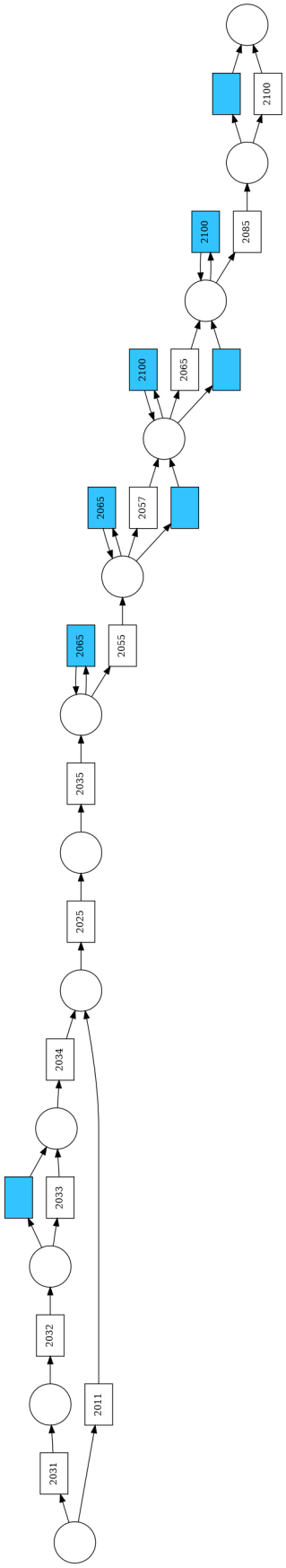
The number of repair steps performed per model are similar to the previous example case A. In particular, the differences between the number of repair steps are similar. If we consider that case A has almost 3 times more variants and more than double number of instances, it might be surprising that the number of adaptions to the original model is at the same level with 11 for both `move-loc` in this example and in case A. Naturally, the `repair-all-traces` model has

most repair steps. Like in case A, this large number of adaptions leads to fitness of all test traces in this case. Moreover, the repair procedure in `kpi-based` and `avoid-flower` works as expected as these two models have equal repair steps.

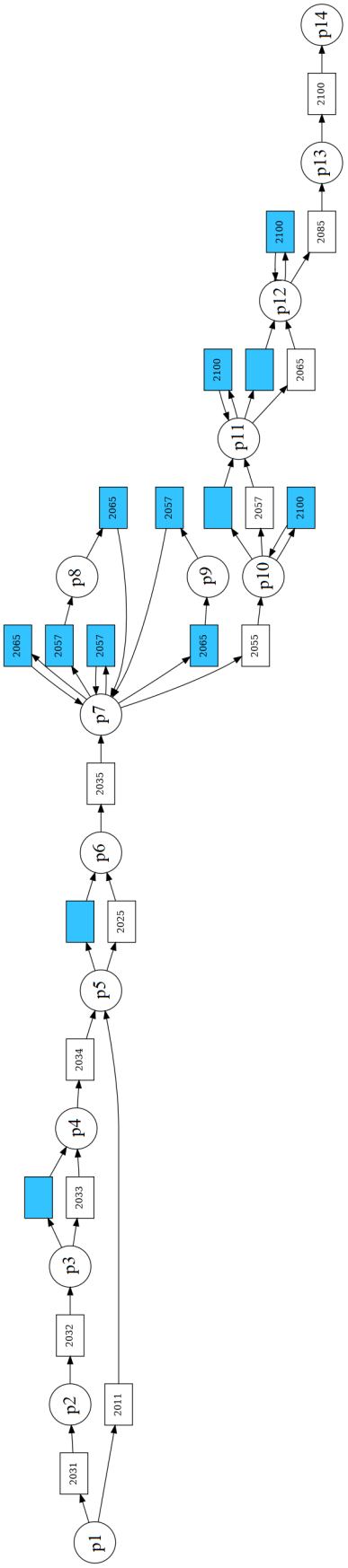


(a) Original model as starting point

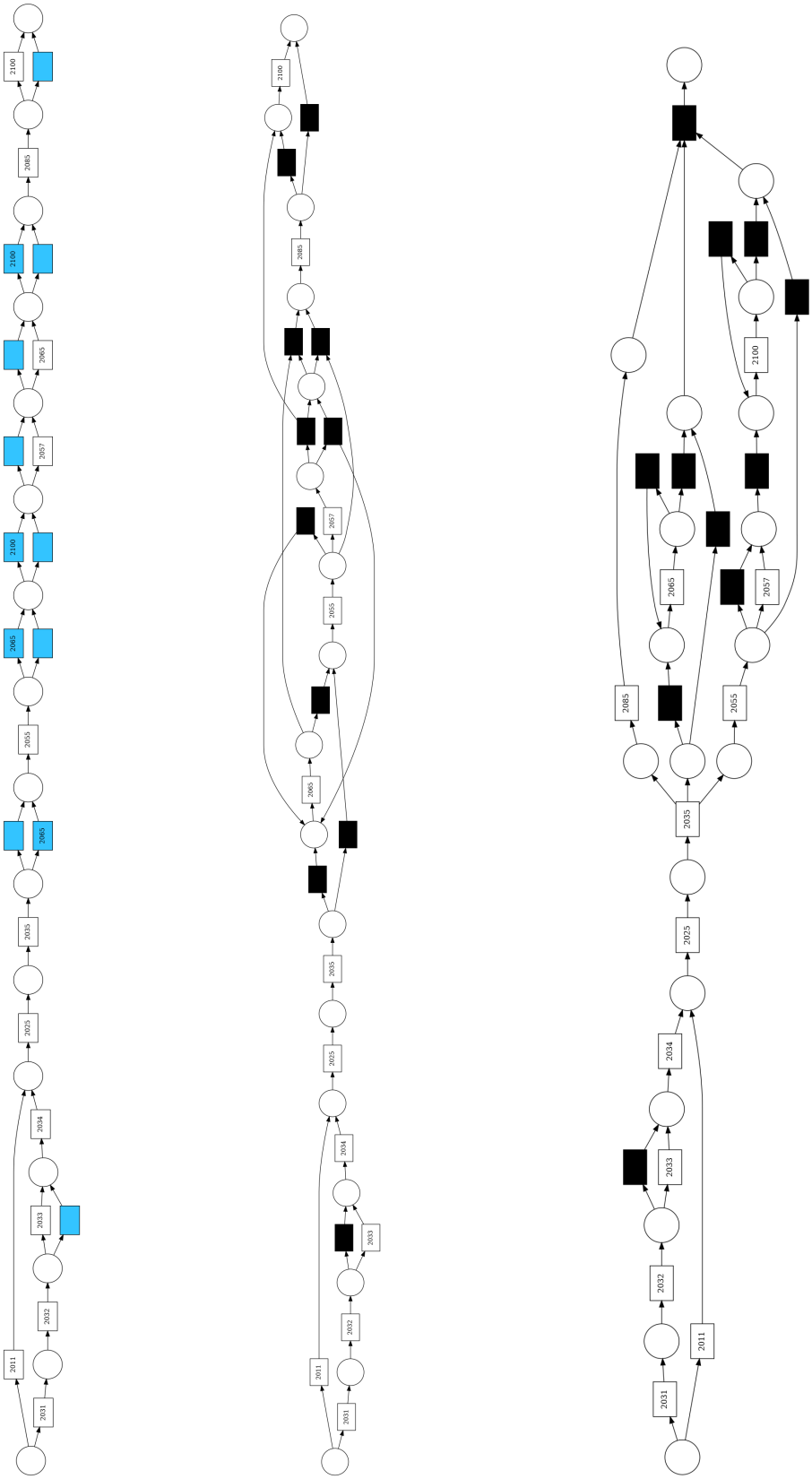
(b) repair-all-traces using Fahland and Van der Aalst technique [20]



(c) KPI-based (following the example by Dees et al.: move as feature [1])



(d) move-loc: move + location as feature



(e) avoid-flower: model integrates same steps as 5.3c but avoids imprecise flower model

(f) repaired-HEU: model discovered by heuristics miner on repaired log

(g) repaired-IM: model discovered by inductive miner on repaired log

Figure 5.3: Comparison between Petri net models for case B

Table 5.3: Characteristics of different models for case B

Model	# Fitting Traces	Mean OEE	Median OEE	Standard deviation of OEE	# Repair Steps	Simplicity	Precision	Generalization	Log Fitness
original	47	1.790	1.818	0.369	0	1.000	0.999	0.798	0.868
kpi-based	113	1.787	1.808	0.420	8	0.667	0.922	0.716	0.964
repair-all-traces	169	1.802	1.810	0.422	18	0.570	0.739	0.656	0.971
move-loc	93	1.807	1.815	0.390	11	0.639	0.891	0.688	0.921
avoid-flower	113	1.787	1.808	0.420	8	0.714	0.939	0.714	0.959
repaired-HEU	47	1.790	1.818	0.369	—	0.667	0.974	0.749	0.936
repaired-IM	164	1.810	1.816	0.411	—	0.706	0.808	0.649	0.998

Wrap-Up

One take-away from this example is that `move-loc` once more produces a satisfying result which can be interpreted as that the extension to use the move and its location as feature makes sense for a smaller example with fewer process instances too. This means that a more thorough split into "good" and "bad" traces which is done by including the location of a move as feature, leads to an improved result. In other words, we can conclude that in this example, the `move-loc` extension manages to optimize the process regarding OEE better than the model `kpi-based` which follows the example by Dees et al. [1]. The result of `move-loc` is only outperformed regarding average OEE by `repaired-IM` and `repaired-HEU` which use process discovery. Furthermore, we see that a small number of adaptions can have an impact regarding how many and which traces fit. Figure 5.3 shows that the different repair steps between `move-loc` and `kpi-based` have an impact in the results. The model `avoid-flower` perform as expected as it fits the same traces as `kpi-based`. For another 17 cases, `avoid-flower` gets the same result as `kpi-based`. Hence it can be assumed that the extension to avoid overly general flower models is well suited for manufacturing processes according to results of these cases. Interestingly, the strict sequence of `avoid-flower` does not harm the fitness to test traces in most examples. One can conclude that the behavior in the log is not very diverse as many traces fit with the behavior that allows to fire transition multiple times in a row as good as a stricter sequence of transitions. Hence, the more flexible behavior as shown in `kpi-based` cannot be replayed by the `avoid-flower` models as they are discussed in this thesis. Nevertheless, this does not make a significant difference in the examples discussed in the scope of this thesis.

5.3 Case Study 2: Administration

Case Description

In addition to the manufacturing process data, the concept of this thesis is also applied on administrative process data. The data used for the analysis of an administrative process is taken from BPI Challenge 2020[2]. This data is publicly available. The topic is business traveling. There are different processes in the data set, such as domestic declarations, international declarations and travel permits (including all related events of relevant prepaid travel cost declarations

[2]van Dongen, B.F., Dataset BPI Challenge 2020. 4TU.Centre for Research Data. https://doi.org/10.4121/uuid:52fb97d4-4588-43c9-9d04-3604d4613b51

and travel declarations). Data preparation and anonymization is already done by the provider Eindhoven University of Technology.

The process for handling various permits and declaration documents, including domestic and international declarations, pre-paid travel costs, and payment requests, follows a standardized flow. Employees submit a request for approval by the travel administration, and if approved, the request proceeds to the budget owner and supervisor. In some instances, director approval may also be required. Subsequently, the trip takes places or a payment request is made and fulfilled. There are two main types of trips—domestic and international. Domestic trips do not require prior permission, allowing employees to undertake them and seek reimbursement afterward. However, international trips require supervisor approval obtained through a travel permit before making any arrangements. Reimbursement claims can be filed either immediately after incurring costs or within two months post-trip for expenses like hotel and food. The data set used in this example focuses on the payment request process flow which ends when payment is fulfilled.

The data for this case study does not only differ from the manufacturing data regarding the topic but also in size. The example log *Request For Payment* contains 6886 cases and 19 distinct events (see table 5.1). The number of cases is far more than what is available for distinct materials (i.e. processes) in manufacturing data. It is interesting to see possible differences between the results of model repair for the payment request process and those results for processes with fewer cases input data. Regarding the number of distinct events, the administrative process is of approximately the same size as the manufacturing processes.

Since there is only few domain knowledge available, the original model is created by incorporating as many cases as possible by using as few process variants as possible. This is the third variant of creating an original model in the description in the previous section. In this case we can achieve a good result as only one variant accounts for 30 % of all cases which is 3011 cases of 6886 in total. The KPI value used here is the execution time. This is compliant to the analysis done in [1] where the time performance of administrative processes is compared between different applications of model repair. Time is measured using a built-in function in PM4Py which returns the duration of each case in seconds.

The threshold is set to the median execution time of all traces which is 704,367.5 seconds (approximately 8 days and 3 hours (8.15 days)). Note that the threshold is used to split cases into "desirable" ones that we want to include to the model and "bad" ones that we do not want to include. The figures shown in the results table are higher than the threshold because they represent average of values of all traces that fit the (repaired) model. For comparison, the mean execution time of all 6886 traces is 12 days (1,037,472.74 seconds).

The output of integrating all test traces into the original model leads to an overwhelming "spaghetti" process model that can be found in figure 5.5. This complex structure of the `repair-all-traces` supports the idea that we need to set the threshold more strictly for this example, otherwise the other repaired models (`kpi-based`, `move-loc` etc.) will look also very complex and one cannot read anything from it anymore. Hence the threshold is set lower than the 25 % quantil in the manufacturing examples. Of course we would need to set the threshold to the 75 % quantil in this case to achieve the same effect since here a lower KPI value is more desirable.

Results

Interestingly, the use of all possible traces for repair does not lead to improved execution time. On the contrary, the mean and median execution time of the `repair-all-traces` model is the worst among all models. However, the number of fitting traces is highest for the `repair-all-traces` model. This is expected and plausible. Furthermore, it is interesting that the `original` model achieves the best mean and median execution time (displayed in subfigure 5.4a).

Table 5.4: Model characteristics for administrative process *Request For Payment* from BPI Challenge 2020

Model	# Fitting Traces	Mean execution time	Median execution time	Std. dev. execution time	# Repair Steps	Simplicity	Precision	Generalization	Log Fitness
original	1002	11.45 d	8.03 d	12.44 d	0	1.000	0.966	0.978	0.938
kpi-based	1772	11.82 d	8.24 d	13.68 d	11	0.545	0.603	0.706	0.958
repair-all-traces	2265	11.74 d	8.23 d	14.44 d	32	0.600	0.215	0.490	0.981
move-loc	1571	11.66 d	8.21 d	12.89 d	10	0.583	0.784	0.694	0.947
avoid-flower	1771	11.70 d	8.24 d	12.67 d	11	0.667	0.735	0.877	0.989
repaired-HEU	0	—	—	—	—	0.581	0.796	0.745	0.805
repaired-IM	1765	11.70 d	8.24 d	12.68 d	—	0.695	0.774	0.879	0.988

Comparison between the model that has been repaired using `kpi-based` following the example by Dees et al. [1] and the model repaired by the extension that uses the location as well, shows that the latter model `move-loc` is has a lower, hence better mean and median execution time. In fact, `move-loc` achieves the best mean and median execution time if we disregard the original model. This is why the respective cells are marked in light green in table 5.4. However, `move-loc` fits fewer test traces than the `kpi-based` model. The two models `move-loc` and `kpi-based` are displayed in in subfigure 5.4c and 5.4b respectively. At first glance, these two models look pretty similar. There are some differences such as that the `kpi-based` contains a loop structure.

The cells marked in slightly darker green in table 5.4 show that the `kpi-based` alongside `avoid-flower` matches most test cases (if we disregard the `repair-all-traces` model). This is not surprising since `kpi-based` and `avoid-flower` are designed to create equal results or at least repair the same moves, albeit in different ways. The `avoid-flower` model can be seen in subfigure 5.4d[3]. When compared to case study 1, the `kpi-based` and `avoid-flower` model fit better than `move-loc` and also slightly better than `repaired-IM`. The model `avoid-flower` shows good values in simplicity, precision and generalization. The `avoid-flower` is already good in these dimensions in case study 1. Since the extension to avoid flower models has been drafted with the intention to achieve better precision than the "classic" `kpi-based` repair, the better values for `avoid-flower` support the idea that this extension has been drafted with. Nevertheless, in this example the scores are only marginally better. This is due to the fact that in all models the number of adaptions is rather low which in turn is caused by the rather strict (low) threshold. Naturally, the number of cases classified as desirable is lower compared to the examples in case study 1 because of this low threshold. Again, note that in this example low means "good" as we refer to execution time as KPI.

The scores for `move-loc` are not only interesting because it has the best average execution time but also because the precision and log fitness are good even though the number of fitting traces is relatively low. The graphic in subfigure 5.4c shows that new transitions are added in different places of the model rather than creating "clustered" structured with transitions going

[3]Note that this image is split into 3 parts and must be read from top left to bottom right. There are no lines added between the first row and the second row of the image for the sake of readability. One can imagine that lines that link the end of the first row to the begin of the second row are there just like for the second row to the third row.

from and to the same place as in `kpi-based`. Note that `move-loc` has one repair step fewer than `kpi-based`. We can conclude that this leads to a shorter average execution time in this example because a more thorough decision between which traces are included, can help to distinguish desirable traces from undesirable ones.

Furthermore, one we can observe that it is possible to apply model repair on an administrative process in a way that a large share of possible behavior is now represented by the models `kpi-based` and `avoid-flower`. Hence the relatively high number of fitting traces for these two models. This could be a hint that there are not many repetitive activities in the log and hence also not in the model as the avoid flower version adds new transitions in a sequence which does not allow for execution in random order. If there are many repetitive transitions that would need to be executed in different places, the `avoid-flower` model would probably show a worse score.

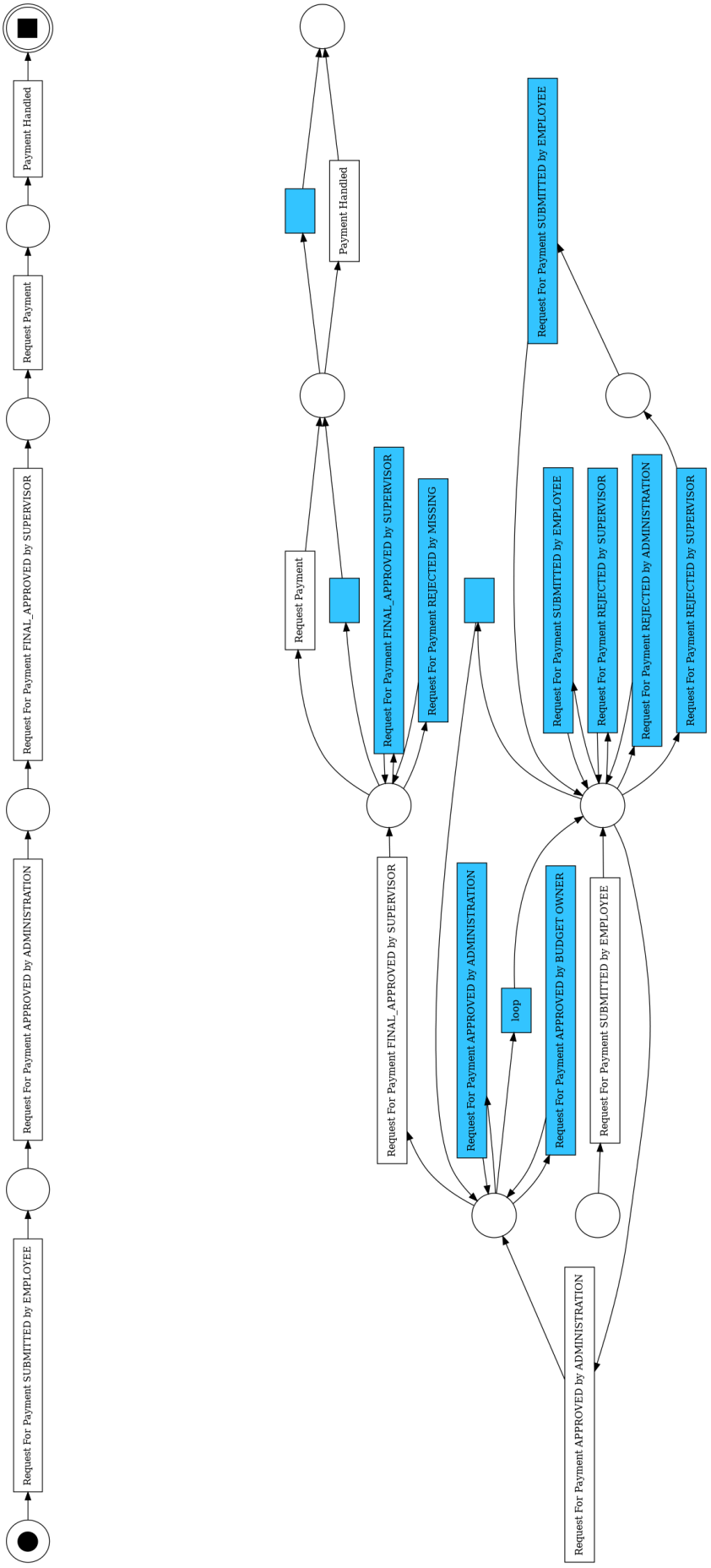
Wrap-Up

Similar to case B in case study 1, the `move-loc` extension leads to improved KPI values but a lower number of fitting traces than the other repaired models (`kpi-based` and `avoid-flower`). One conclusion can be that the differences between manufacturing and administrative data is very large hence the model repair extensions proposed in this thesis are applicable in both fields.

Interestingly, the mean execution time of `kpi-based` is 0.12 days (approximately 3 hours) more than the mean for `avoid-flower`, even though `kpi-based` fits only one more trace. This single extra trace is the reason why the mean time differs while the median time is equal. This shows that not only the limited size of the log must be kept in mind but also that single trace can have a large impact. Note that case study 2 is added to this thesis for comparison between manufacturing and administrative processes. The selected process *Request For Payment* is a rather small one among the BPI Challenge 2020 data as the focus is on readable and understandable process models.

On the other hand, the relatively bad performance of the discovered models `repaired-IM` and `repaired-HEU` supports the argument that this process has no large "main street" of frequent behavior that covers a large percentage of behavior. While the heuristic model does not fit any of the testing traces, the inductive miner discovers a model that has a high generalization. Even though, precision is low, this means that there are only few parts of the model that are rarely visited which suggests an efficient model. The reason for the relatively bad precision score could be that there are some parts of the model that represent behavior that is not in the testing log, although these parts are not visited often according to the testing log.

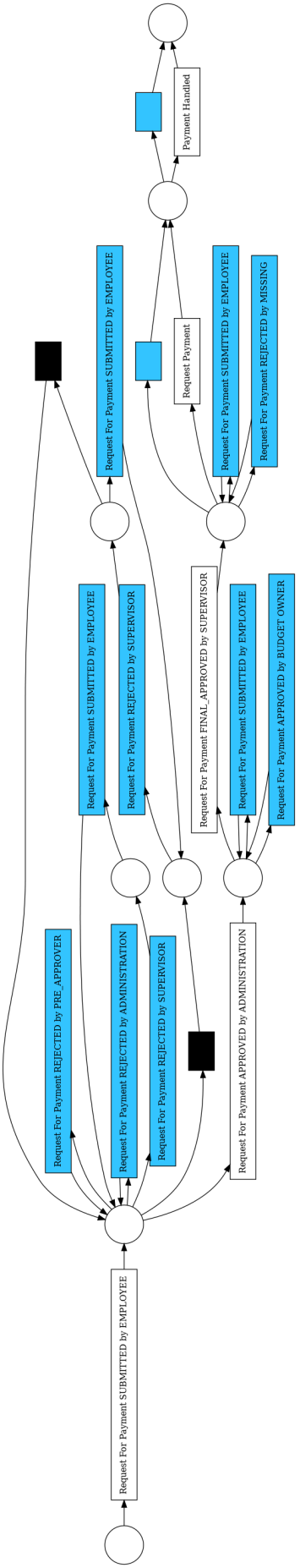
Note that none of the new models outperforms the original model regarding mean and median execution time. Furthermore, among the repaired models, only `move-loc` has better average execution time time than if all traces are repaired. The good performance of `move-loc` regarding the median execution time supports the idea that this extension is sensible also for administrative use cases even if it only fits few test traces in this example. The good precision emphasizes that the adapted model repair in the form of `move-loc` can sufficiently reconstruct the process. This can serve as the basis for further use of this extension as it balances fitness and complexity based on its more sophisticated classification into good and bad traces during the learning phase of the decision tree.



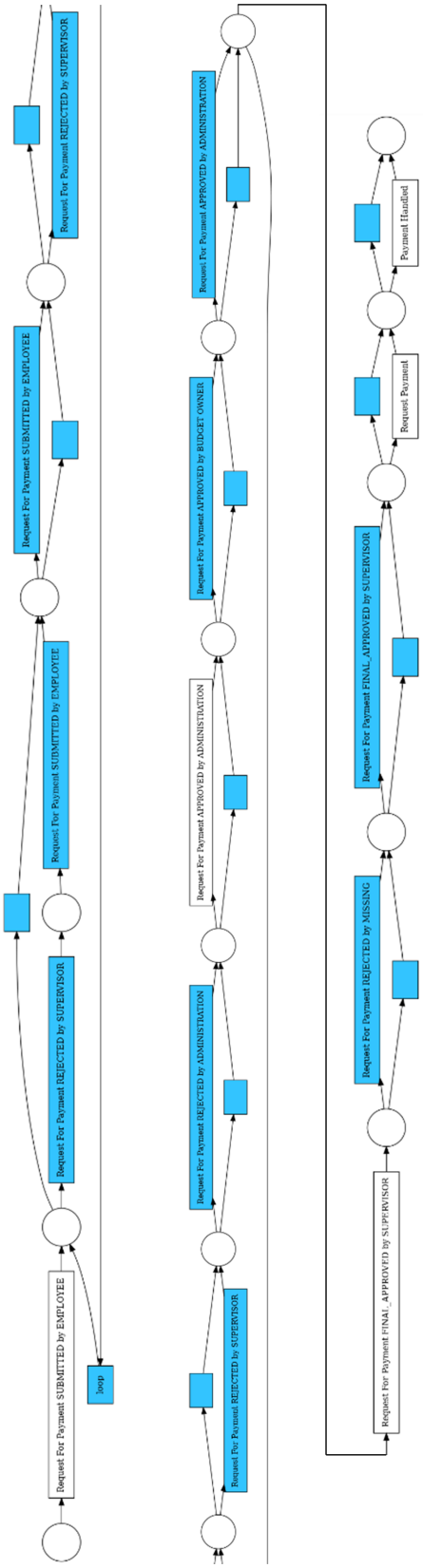
(a) Original model as starting point

(b) KPI-based (following the example by Dees et al.: move as feature [1])

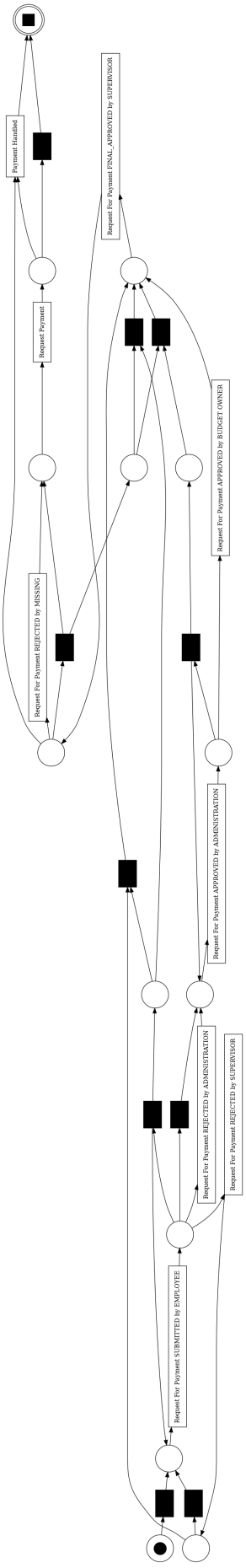
Figure 5.4: Repaired vs. discovered administrative process



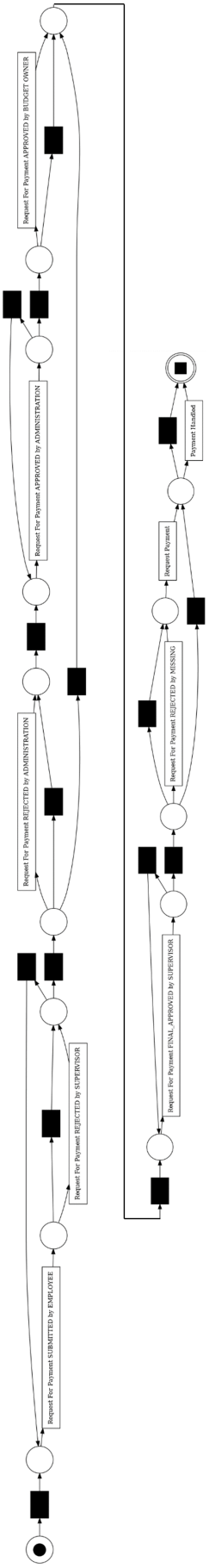
(c) move-loc: move + location as feature



(d) avoid-flower: model integrates same steps as 5.4b but avoids imprecise flower model



(e) repaired-HEU: model discovered by heuristics miner on repaired log



(f) repaired-IM: model discovered by inductive miner on repaired log

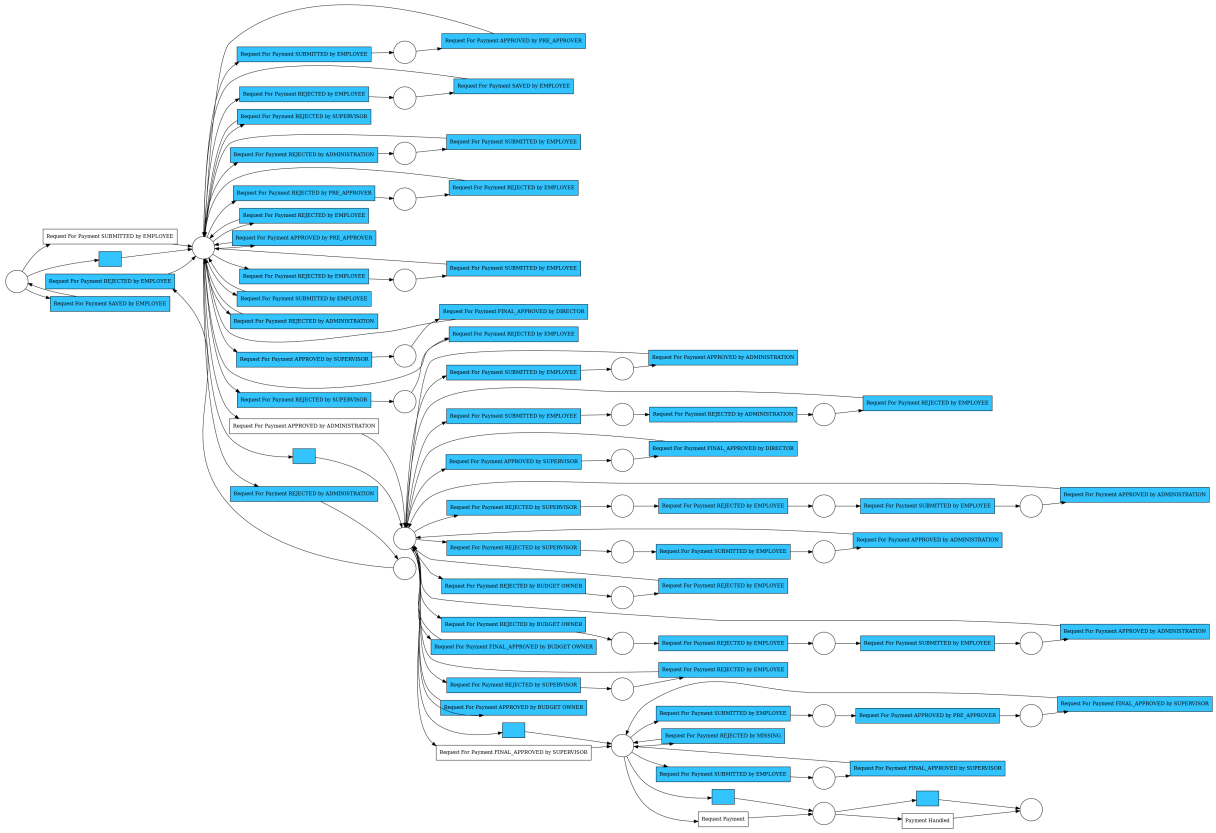


Figure 5.5: Model for process *Request For Payment* from BPI Challenge 2020 - repaired with all test traces

6 Conclusion and Outlook

The aim of this thesis is to apply and compare different versions of model repair, especially extensions to the technique used in the work by Dees et al. [1] as well as Fahland and Van der Aalst [20]. The various KPI-based model repair procedures are conducted on manufacturing data and administrative process data for comparison. Model repair considers alternative traces that are not yet in the model and might be added based on certain criteria. One advantage is that these alternative traces represent valid process behavior as the traces have been tracked in the real world. Hence the existing model is enriched with new behavior that is by default compliant with the real-world process as the new behavior is taken from real world.

The case study conducted in this thesis shows that model repair is applicable on manufacturing process data with the aim to improve certain target KPIs. The result is based on 13 inspected cases of appropriate size. In particular, we showed that models that use certain traces and hence certain alignment steps for repair have better average KPI values than if all possible repair steps are carried out. This is an important finding as it proves that "good" and "bad" traces can be identified using a simple learning method such as decision tree. The good traces can subsequently be used for model repair in a beneficial way. Several KPIs that are commonly used in either manufacturing industry or process mining have been discussed. Overall Equipment Effectiveness is selected as KPI for analysis of manufacturing processes because it is a comprehensive metric that comprises resource availability, output quality and production performance. Hence, it is possible to gain a insightful overview of the quality of single traces using OEE.

In addition to the kind of KPI used, the way it is applied is also worth mentioning. We use mean OEE values throughout the thesis, i.e. the mean OEE of the operations is considered as the OEE of the trace. At this point traces with outlying OEE are removed. For the manufacturing data, the 25 % quantil OEE of the traces is in turn used as a threshold to separate desirable from "bad" traces for creating the decision tree. This setting to mean value per trace is used for practical reasons. It is a simple and sensible way to map the measurements from the activity-level to the trace-level. It is necessary to calculate the OEE per trace because the intended KPI value is required to be available as attribute on trace level.

The comparison between different repaired models shows that the selection of traces with good KPI values has better outcome than if all traces are used for repair. Moreover, the impact of the various adapted repair techniques is discussed throughout the case study chapter. The extensions to use an adapted way of integrating new transitions to get more precise models (`avoid-flower`) and to add the location of a move to the feature for learning the pattern (`move-loc`) are new contributions in the field of KPI-based model repair. The use of the `avoid-flower` version makes sense if the process is less diverse, i.e. if there are fewer variants and it does not cause problems if the mostly used transitions are arranged as a sequence instead of adding them as transitions which can fire at any time as they are added to the same place as input and output. In the case study avoid-flower is applicable for most examples of manufacturing data as the sequential structure fits as many test traces as the more flexible structure created by `kpi-based`. Furthermore, it is also applicable in the example of administrative process because there the tasks are not repetitive which makes the repaired avoid-flower compliant to the log.

The results show that there are certain processes for which it makes sense to include the location to the move as a feature. This is particularly sensible if the data contains many different traces that lead to different outcome for the same transition in different places. In fact, case study 1 shows that `move-loc` models can keep up with the example by Dees et al. [1] (model: `kpi-based`) in both cases of manufacturing processes. This is because the more thorough distinction between good and bad traces leads to an improved model repair. Interestingly, for 8 cases inspected, the

number of repair steps is higher if the move-loc extension is applied than for kpi-based models.

Various measurements for model quality dimensions have been used to assess the repaired models. The results show that the version to avoid flower models also manages to retain a simple model regarding precision and generalization. On the other hand, comparison with a model that is discovered from the repaired log shows that the newly discovered model can be more similar to the original model that the repaired model is. Interestingly, a repaired model, which is created using a model repair technique, is but an adapted version of the original and would hence be expected to be more similar to the original. In general, the discovered models by inductive miner and heuristics miner show better values in the quality dimensions compared to the repaired models in many cases. However, if the original model already fits lots of traces, newly discovered models can hardly keep up with the number of fitting traces. Furthermore, the main target KPI to be optimized, namely the mean OEE of the fitting traces, is better for the discovered models for some examples.

The number of fitting test traces and the log fitness scores are ways to check the fitness of the model to the log. They provide a good overview of the model characteristics as fitness is - alongside with precision - the most commonly used quality dimension. Although precision is almost always better for discovered models, as the algorithms are built to consider precision, generalization is at a similar level for repaired and discovered models. Apart from the comparison between discovered and repaired models, it can be said that the different model repair versions, such as the new extensions avoid-flower, use of move-loc and the example by Dees et al. result in similar values regarding model quality dimensions. The differences lies rather in the results for the fitting traces from the testing data and the mean OEE value of these unseen traces.

Administrative processes have been analyzed as well in order to complement the case studies and to point out possible differences between repairing administrative and manufacturing processes. Administrative processes rather allow to use a original model that is built from the most common variants. One effect of this is that it is possible to incrementally add new behavior to the original model so that more complexity and the hoped-for goal, i.e. reduced mean throughput time, can be balanced. The comparison between the discovered and repaired models for the administrative process shows that a good original model to start with also helps the repair procedure to keep up the good KPI value and at the same time keep the repaired model close to the original. Due to the relative few activities added to the process in the example Request For Payment Log, the avoid-flower variant leads to a decent result regarding trace fitness. In case study 2, the repaired models in the administrative process require fewer flexibility than the models in case study 1, as the threshold is set more strictly. This leads to that we only look at certain (most frequent) parts of the process. Consequently, the performance is good because few traces can make a large difference to average KPI values and the applied algorithms are able to find the pattern between trace behavior and KPI value in the log that is to be added to the model.

Regarding the use of discovery on repaired log, we can conclude that it makes sense to apply the Inductive or heuristics miner on repaired log if there is not much domain knowledge available and we want to repair the model in a way that it fits test log. This is especially true for models discovered with inductive miner. However, the discovered models have very few in common with the original model. Hence it is especially sensible to use discovery instead of model repair in cases where it is difficult to start with an original model that was designed by a domain expert.

Naturally, the original model has a large influence on the outcome of model repair as discussed in section 5.1.3. The original model will typically be created by a domain expert or using the knowledge of such experts to represent the process as it is designed to operate. In this thesis, original process models are created by discovering most frequent behavior from the log. As there is some knowledge about the structure of manufacturing data, we can exclude rework steps and

set the activities in the order as they are planned. However, in most cases the reference model is created by discovery. It is interesting do further work on model repair with consulting process managers from the respective organizations to create original models to start with.

Only few processes from manufacturing industry were actually applicable for model repair. The reason is that - in the data used in this thesis - many processes either consist of few instances (orders) or few activities (operations). In order to leverage process model repair for insights and improvements on real world processes, domain knowledge is required. For this thesis, only basic domain knowledge is available which allows e.g. to create original models in the sequence of operations as they are planned and excluding rework activities. Nevertheless, in order to choose a wider range of suitable processes and to make more specific statements about the effect of model repair, even more knowledge about the underlying manufacturing procedures is necessary. Moreover, the lack of processes with many instances is the reason why we use the 25 % quantil of OEE as threshold. If we would be able to use a stricter threshold, we might be able to focus even more on the distinction between desirable and undesirable traces which could lead to an even better result due to a better learning of the underlying pattern between the trace behavior and KPI value within the log.

Results of model repair depend on the way the alignments are calculated. Two different algorithms applied on the same log and model can result in different alignments. Therefore, the choice of the algorithm to compute alignments and hence possible repair steps is as important as the choice of the original model. As mentioned, alignments between model and log are calculated using the Dijkstra Less Memory algorithm in this thesis [44, 43]. The versions of model repair as they are presented in this thesis show a pattern of characteristics of different versions such as the `avoid-flower` and `move-loc`. The case study chapter mainly focuses on these patterns that represent a wider view on the topic and is not necessarily limited to the concrete examples that are discussed.

6.1 Outlook

One of the main research fields of this thesis is to apply different versions of model repair. There are even more possible ways to adapt model repair in order to use it tailored for a certain business domain or for a special kind of process data (e.g. few distinct activities). Versions of model repair can also focus on structural adaptions of repair steps. There are possibilities to improve model repair measures by focusing on simpler structures which represent same or at least similar behavior. One approach in this regard has been proposed in this thesis, in the form of the new extension avoid-flower. Since we have seen that the applicability of this version is promising, we propose to carry out more research in the field of model repair versions, especially even more structural improvements.

For the second extension proposed in this thesis, namely use of move including the location as feature, we proved that it is suitable to apply it when a more complex pattern should be identified. The case studies showed that an adaption of the feature already makes a difference in the model repair procedure. When even more information about a process is available, an even higher number of features and more detailed features can be used. This can lead to a more precise model repair that better represents the pattern between trace characteristics (such as which deviations to the reference model there are) and the KPI target value per trace.

As mentioned in the conclusion section above, this thesis only considers OEE as KPI. Further work could be done with focus on certain parts of OEE or even other, more specialized, KPIs. This can also comprise the use of wider KPIs that are not specific to the field of manufacturing industry as in this use case. Furthermore, it is advisable to structure the data in way that the target KPI values are available as trace attribute without the need to use mean values. A related

but different approach in this context is to use single transitions (i.e. activities) for model repair instead of transitions from traces. This would mean that a transition is added if the represented activity is predicted to account for a desirable KPI value. Of course this separation of activities and traces might cause issues, such as logical inconsistencies in the process execution or even more frequently harmful repair suggestions, such as the removal of workplace safety measures.

Other machine learning techniques and algorithms might also be a way to improve model repair as they might be better in finding the link between features and target values. Further work with techniques such as deep learning can be done to see whether the results can be further improved. One precondition is that sufficiently large data sets from suitable domains are available. In particular, large process data sets combined with fundamental domain knowledge can be the foundation for more in-depth research in the field of KPI-based model repair.

Bibliography

[1] Marcus Dees, Massimiliano de Leoni, and Felix Mannhardt. Enhancing Process Models to Improve Business Performance: A Methodology and Case Studies. In Hervé Panetto, Christophe Debruyne, Walid Gaaloul, Mike Papazoglou, Adrian Paschke, Claudio Agostino Ardagna, and Robert Meersman, editors, *On the Move to Meaningful Internet Systems. OTM 2017 Conferences*, Lecture Notes in Computer Science, pages 232–251, Cham, 2017. Springer International Publishing.

[2] ISO 13053 Quantitative methods in process improvement — Six Sigma, 2011.

[3] David Parmenter. *Key performance indicators: developing, implementing, and using winning KPIs*. John Wiley & Sons, 2015.

[4] ISO 22400 Automation systems and integration — Key performance indicators (KPIs) for manufacturing operations management, 2014.

[5] Yoshiro Fukuda and Robert Patzke. Standardization of Key Performance Indecator for manufacturing execution system. In *Proceedings of SICE Annual Conference 2010*, pages 263–265, August 2010.

[6] Wil van der Aalst. *Process Mining - Data Science in Action*, volume 2. Springer, 2016.

[7] Wil van der Aalst. *Process mining : discovery, conformance and enhancement of business processes*. Springer, Berlin [u.a.], 2011.

[8] Michele Chinosi and Alberto Trombetta. Bpmn: An introduction to the standard. *Computer Standards & Interfaces*, 34(1):124–134, 2012.

[9] Murata Tadao. Petri nets: properties, analysis and applications. *Proceedings of the IEEE*, 77(4), 1990.

[10] Wil Van Der Aalst, Arya Adriansyah, Ana Karla Alves De Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, Jagadeesh Chandra Bose, Peter Van Den Brand, Ronald Brandtjen, Joos Buijs, et al. Process mining manifesto. In *Business Process Management Workshops: BPM 2011 International Workshops, Clermont-Ferrand, France, August 29, 2011, Revised Selected Papers, Part I 9*, pages 169–194. Springer, 2012.

[11] Wil MP van der Aalst. Foundations of process discovery. In *Process Mining Handbook*, pages 37–75. Springer, 2022.

[12] Josep Carmona, Boudewijn van Dongen, Andreas Solti, and Matthias Weidlich. Conformance checking. *Switzerland: Springer.[Google Scholar]*, 2018.

[13] Massimiliano de Leoni. Foundations of Process Enhancement. In Wil M. P. van der Aalst and Josep Carmona, editors, *Process Mining Handbook*, Lecture Notes in Business Information Processing, pages 243–273. Springer International Publishing, Cham, 2022.

[14] Wil Van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE transactions on knowledge and data engineering*, 16(9):1128–1142, 2004.

[15] A. J. M. M. Weijters, Wil M. P. van der Aalst, and A. K. Alves de Medeiros. Process mining with the heuristics miner-algorithm. *Technische Universität Eindhoven, Tech. Rep. WP*, 166(July 2017):1–34, 2006.

[16] Esmita P Gupta. Process mining a comparative study. *International Journal of Advanced Research in Computer and Communications Engineering*, 3(11):5, 2014.

[17] Ana Karla A de Medeiros, Anton JMM Weijters, and Wil MP van der Aalst. Genetic process

mining: an experimental evaluation. *Data mining and knowledge discovery*, 14(2):245–304, 2007.

[18] Oliver Kopp, Daniel Martin, Daniel Wutke, and Frank Leyman. The difference between graph-based and block-structured business process modelling languages. *Enterprise Modelling and Information Systems Architectures (EMISAJ)*, 4(1):3–13, 2009.

[19] Sander JJ Leemans, Dirk Fahland, and Wil MP van der Aalst. Scalable process discovery with guarantees. In *Enterprise, Business-Process and Information Systems Modeling: 16th International Conference, BPMDS 2015, 20th International Conference, EMMSAD 2015, Held at CAiSE 2015, Stockholm, Sweden, June 8-9, 2015, Proceedings*, pages 85–101. Springer, 2015.

[20] Dirk Fahland and Wil M. P. van der Aalst. Model repair — aligning process models to reality. *Information Systems*, 47:220–243, January 2015.

[21] S. J. van Zelst, B. F. van Dongen, W. M. P. van der Aalst, and H. M. W. Verbeek. Discovering workflow nets using integer linear programming. *Computing*, 100(5):529–556, May 2018.

[22] Jorg Desel and Javier Esparza. *Free choice Petri nets*, volume 40. Cambridge university press, 1995.

[23] Wolfgang Reisig and Grzegorz Rozenberg. *Lectures on petri nets i: basic models: advances in petri nets*, volume 1491. Springer Science & Business Media, 1998.

[24] James L. Peterson. Petri nets. *ACM Comput. Surv.*, 9(3):223–252, sep 1977.

[25] Wil MP Van der Aalst. Verification of workflow nets. In *International Conference on Application and Theory of Petri Nets*, pages 407–426. Springer, 1997.

[26] Wil M.P. van der Aalst and Kees M. van Hee. Workflow management: Models, methods, and systems. In *International Conference on Cooperative Information Systems*, 2002.

[27] Jörg Desel. Process modeling using petri nets. *Process-Aware Information Systems: Bridging People and Software through Process Technology*, pages 147–177, 2005.

[28] Mathias Weske. *Business process management architectures*. Springer, 2007.

[29] J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. A genetic algorithm for discovering process trees. In *2012 IEEE Congress on Evolutionary Computation*, pages 1–8, June 2012. ISSN: 1941-0026.

[30] Joos CAM Buijs, M La Rosa, Hajo A Reijers, Boudewijn F van Dongen, and Wil MP van der Aalst. Improving business process models using observed behavior. In *International Symposium on Data-Driven Process Discovery and Analysis*, pages 44–59. Springer, 2012.

[31] Wil MP van der Aalst. Process mining: a 360 degree overview. In *Process Mining Handbook*, pages 3–34. Springer, 2022.

[32] A. Rozinat and W.M.P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95, 2008.

[33] Joos CAM Buijs, Boudewijn F van Dongen, and Wil MP van der Aalst. Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity. *International Journal of Cooperative Information Systems*, 23(01):1440001, 2014.

[34] Alessandro Berti and Wil MP van der Aalst. A novel token-based replay technique to speed up conformance checking and process enhancement. In *Transactions on Petri Nets and Other Models of Concurrency XV*, pages 1–26. Springer, 2021.

[35] Joos CAM Buijs, Boudewijn F Van Dongen, Wil MP van Der Aalst, et al. On the role of fitness, precision, generalization and simplicity in process discovery. In *OTM Conferences (1)*, volume 7565, pages 305–322, 2012.

[36] Jorge Munoz-Gama and Josep Carmona. A fresh look at precision in process conformance. In *International Conference on Business Process Management*, volume 6336, pages 211–226, 09 2010.

[37] Jorge Munoz-Gama and Josep Carmona. Enhancing precision in process conformance: Stability, confidence and severity. In *2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pages 184–191. IEEE, 2011.

[38] Arya Adriansyah, Jorge Munoz-Gama, Josep Carmona, Boudewijn F Van Dongen, and Wil MP Van Der Aalst. Measuring precision of modeled behavior. *Information systems and e-Business Management*, 13:37–67, 2015.

[39] Wil van der Aalst, Arya Adriansyah, and Boudewijn van Dongen. Replaying history on process models for conformance checking and performance analysis. *WIREs Data Mining and Knowledge Discovery*, 2(2):182–192, 2012.

[40] Borja Vázquez-Barreiros, Manuel Mucientes, and Manuel Lama. Prodigen: Mining complete, precise and minimal structure process models with a genetic algorithm. *Information Sciences*, 294:315–333, 2015.

[41] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on computing*, 18(6):1245–1262, 1989.

[42] Massimiliano De Leoni, Wil MP Van Der Aalst, and Boudewijn F Van Dongen. Data- and resource-aware conformance checking of business processes. In *Business Information Systems: 15th International Conference, BIS 2012, Vilnius, Lithuania, May 21-23, 2012. Proceedings 15*, pages 48–59. Springer, 2012.

[43] Arya Adriansyah, Boudewijn F van Dongen, and Wil MP van der Aalst. Conformance checking using cost-based fitness analysis. In *2011 ieee 15th international enterprise distributed object computing conference*, pages 55–64. IEEE, 2011.

[44] Mathilde Boltenhagen. *Process Instance Clustering Based on Conformance Checking Artefacts*. PhD thesis, Université Paris-Saclay, 10 2021.

[45] HongDa Qi, YuYue Du, Liang Qi, and Lu Wang. An approach to repair petri net-based process models with choice structures. *Enterprise Information Systems*, 12(8-9):1149–1179, 2018.

[46] Chen Li, Manfred Reichert, and Andreas Wombacher. Discovering reference models by mining process variants using a heuristic approach. In *International Conference on Business Process Management*, pages 344–362. Springer, 2009.

[47] Renata Gomes Frutuoso Braz, Luiz Felipe Scavarda, and Roberto Antonio Martins. Reviewing and improving performance measurement systems: An action research. *International Journal of Production Economics*, 133(2):751–760, 2011.

[48] Carl-Fredrik Lindberg, SieTing Tan, JinYue Yan, and Fredrik Starfelt. Key Performance Indicators Improve Industrial Performance. *Energy Procedia*, 75:1785–1790, 2015.

[49] Marlon Dumas, Marcello La Rosa, Jan Mendling, Hajo A Reijers, et al. *Fundamentals of business process management*, volume 1. Springer, 2013.

[50] Kanako Tanaka. Assessment of energy efficiency performance measures in industry and their application for policy. *Energy policy*, 36(8):2887–2902, 2008.

[51] Peter Muchiri and Liliane Pintelon. Performance measurement using overall equipment effectiveness (oee): literature review and practical application discussion. *International journal of production research*, 46(13):3517–3535, 2008.

[52] Rafael Lorenz, Julian Senoner, Wilfried Sihn, and Torbjørn Netland. Using process mining to improve productivity in make-to-stock manufacturing. *International Journal of Production Research*, 59(16):4869–4880, 2021.

[53] Margret Bauer, Matthieu Lucke, Charlotta Johnsson, Iiro Harjunkoski, and Jan C. Schlake. Kpis as the interface between scheduling and control. *IFAC-PapersOnLine*, 49(7):687–692, 2016. 11th IFAC Symposium on Dynamics and Control of Process SystemsIncluding Biosystems DYCOPS-CAB 2016.

[54] Jorge Arinez, Stephan Biller, Kevin Lyons, Swee Leong, Goudong Shao, Byeong Eon Lee, and John Michaloski. Benchmarking production system, process energy, and facility energy performance using a systems approach. In *Proceedings of the 10th performance metrics for intelligent systems workshop*, pages 88–96, 2010.

[55] Florian Johannsen, Susanne Leist, and Gregor Zellner. Six sigma as a business process management method in services: analysis of the key application problems. *Information systems and e-business management*, 9(3):307–332, 2011.

[56] Sanjit Ray and Boby John. Lean Six-Sigma application in business process outsourced organization. *International journal of lean six sigma*, 2(4):371–380, 2011.

[57] Gerald M Taylor. *Lean six sigma service excellence: A guide to green belt certification and bottom line improvement*. J. Ross Publishing, 2008.

[58] Marcello Colledani, Tullio Tolio, Anath Fischer, Benoit Iung, Gisela Lanza, Robert Schmitt, and József Váncza. Design and management of manufacturing systems for production quality. *CIRP Annals*, 63(2):773–796, January 2014.

[59] Kimberly P. Ellis, Russell D. Meller, Joseph H. Wilck IV, Pratik J. Parikh, and Franky Marchand. Effective material flow at an assembly facility. *International Journal of Production Research*, 48(23):7195–7217, 2010.

[60] Artem Polyvyanyy, Wil MP Van Der Aalst, Arthur HM Ter Hofstede, and Moe T Wynn. Impact-driven process model repair. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 25(4):1–60, 2016.

[61] Shazia Sadiq, Wasim Sadiq, and Maria Orlowska. Pockets of flexibility in workflow specification. In *International Conference on Conceptual Modeling*, pages 513–526. Springer, 2001.

[62] Mauro Gambini, Marcello La Rosa, Sara Migliorini, and Arthur HM Ter Hofstede. Automated error correction of business process models. In *International Conference on Business Process Management*, pages 148–165. Springer, 2011.

[63] Niels Lohmann. Correcting deadlocking service choreographies using a simulation-based graph edit distance. In *International Conference on Business Process Management*, pages 132–147. Springer, 2008.

[64] D.M.M. Schunselaar. *Configurable process trees : elicitation, analysis, and enactment*. Phd Thesis 1 (Research TU/e / Graduation TU/e), Technische Universiteit Eindhoven, Eindhoven, October 2016. ISBN: 9789038641263.

[65] Nuttapong Saelim, Parham Porouhan, and Wichian Premchaiswadi. Improving organizational process of a hospital through petri-net based repair models. In *2016 14th International Conference on ICT and Knowledge Engineering (ICT&KE)*, pages 109–115. IEEE, 2016.

[66] Abel Armas Cervantes, Nick RTP van Beest, Marcello La Rosa, Marlon Dumas, and Luciano García-Bañuelos. Interactive and incremental business process model repair. In *On the Move to Meaningful Internet Systems. OTM 2017 Conferences: Confederated International Conferences: CoopIS, C&TC, and ODBASE 2017, Rhodes, Greece, October 23-27, 2017, Proceedings, Part I*, pages 53–74. Springer, 2017.

[67] Giovanni Lugaresi, Antonio Davide Ciappina, Monica Rossi, and Andrea Matta. Exploiting a combined process mining approach to enhance the discovery and analysis of support

processes in manufacturing. *International Journal of Computer Integrated Manufacturing*, 36(2):169–189, 2023.

[68] Shabnam Shahzadi, Xianwen Fang, Usman Shahzad, Ishfaq Ahmad, and Troon Benedict. Research article repairing event logs to enhance the performance of a process mining model. *Mathematical Problems in Engineering*, 2022.

[69] Andreas Rogge-Solti, Ronny S Mans, Wil MP van der Aalst, and Mathias Weske. Improving documentation by repairing event logs. In *The Practice of Enterprise Modeling: 6th IFIP WG 8.1 Working Conference, PoEM 2013, Riga, Latvia, November 6-7, 2013, Proceedings 6*, pages 129–144. Springer, 2013.

[70] Massimiliano De Leoni and Wil MP Van Der Aalst. Aligning event logs and process models for multi-perspective conformance checking: An approach based on integer linear programming. In *Business Process Management: 11th International Conference, BPM 2013, Beijing, China, August 26-30, 2013. Proceedings*, pages 113–129. Springer, 2013.

[71] Wei Song, Xiaoxu Xia, Hans-Arno Jacobsen, Pengcheng Zhang, and Hao Hu. Efficient alignment between event logs and process models. *IEEE Transactions on Services Computing*, 10(1):136–149, 2016.

[72] Jihen Cherni, Ricardo Martinho, and Sonia Ayachi Ghannouchi. Towards improving business processes based on preconfigured kpi target values, process mining and redesign patterns. *Procedia Computer Science*, 164:279–284, 2019.

[73] Branimir Wetzstein, Philipp Leitner, Florian Rosenberg, Ivona Brandic, Schahram Dustdar, and Frank Leymann. Monitoring and analyzing influential factors of business process performance. In *2009 IEEE International Enterprise Distributed Object Computing Conference*, pages 141–150. IEEE, 2009.

[74] Antonello Calabrò, Francesca Lonetti, and Eda Marchetti. Kpi evaluation of the business process execution through event monitoring activity. In *2015 International Conference on Enterprise Systems (ES)*, pages 169–176. IEEE, 2015.

[75] Antonello Calabro, Francesca Lonetti, and Eda Marchetti. Monitoring of business process execution based on performance indicators. In *2015 41st Euromicro Conference on Software Engineering and Advanced Applications*, pages 255–258. IEEE, 2015.

[76] Massimiliano de Leoni, Wil M.P. van der Aalst, and Marcus Dees. A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. *Information Systems*, 56:235–257, 2016.

[77] Alessandro Berti, Sebastiaan van Zelst, and Daniel Schuster. Pm4py: A process mining library for python. *Software Impacts*, 17:100556, 2023.

[78] Alessandro Berti, Sebastiaan J Van Zelst, and Wil van der Aalst. Process mining for python (pm4py): bridging the gap between process-and data science. *arXiv preprint arXiv:1905.06169*, 2019.

[79] Sander JJ Leemans, James M McGree, Artem Polyvyanyy, and Arthur HM ter Hofstede. Statistical tests and association measures for business processes. *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[80] Boudewijn F Van Dongen, Ana Karla A de Medeiros, HMW Verbeek, AJMM Weijters, and Wil MP van Der Aalst. The prom framework: A new era in process mining tool support. In *Applications and Theory of Petri Nets 2005: 26th International Conference, ICATPN 2005, Miami, USA, June 20-25, 2005. Proceedings 26*, pages 444–454. Springer, 2005.