University of Passau

Department of Informatics and Mathematics

Bachelor Thesis

Evaluating the Influence of Feature Interactions on the Learnability of Non-functional Properties

Author:

Jakob Hoffmann

June 25, 2018

Advisors:

Prof. Dr. Sven Apel

Chair of Software Engineering I

Alexander Grebhahn

Chair of Software Engineering I

Hoffmann, Jakob:
Evaluating the Influence of Feature Interactions on the Learnability of Non-functional Properties
Bachelor Thesis, University of Passau, 2018.

Abstract

As many modern software systems are highly configurable, there is the need to gain insights of the influences on the performance of those configuration options. As the configuration space can be of possibly exponential size, this can be solved by employing machine learning techniques to identify those influences on the performance. This approach is implemented in a tool called *SPL Conqueror*. This thesis aims at identifying influences on this machine learning approach as well as finding potential obstacles to learning a software system. To this end, we use realistically generated data with varying characteristics that enable us to identify influences on the learning process in a way that is not possible with real world measurements. An experiment design is proposed with the objective to identify those influences. Additionally, some extensions to the tool used for the data generation named *Thor* had to be made to be able to conduct those experiments. Eventually, the observed results are discussed and the identified influences on the learnability of a software system are outlined.

Contents

List of Figures

List of Tables

List of Code Listings

1. Introduction

As many modern software systems are configurable to a vast extent, end-users can feel overwhelmed by the sheer amount of configuration options [XJF+15]. Those options can not only have huge influences on the desired functionality, but also on the non-functional properties of a software system such as runtime, main memory consumption or binary size. Thus, many end-users opt to use a default configuration or rarely change a single option. This leads to a severe waste of optimization potential built into many software systems that remains unused as identifying the optimal configuration for a certain functionality is difficult. An approach proposed by Siegmund et al. [SGAK15] intents to solve this issue via a machine learning approach, that aims to identify options or interactions between options with the largest influence on the performance. To this end, a *performance-influence model* is learned which – if learned correctly – describes all influences on the performance of a software system in a human readable manner. This approach is implemented in a tool called *SPL Conqueror*. In previous work, this machine learning approach has been found to be effective in finding those influences for real world systems of multiple domains, written in multiple programming languages and for varying configuration option types [SGAK15, GSKA16]. However, those evaluations were only capable of examining that a specific software system is learnable, but not which properties of the system influence this learnability.
According to our understanding, learnability is the capability to learn a *performance-influence model* that correctly contains all influences on the performance. Knowing influences on the learnability enables us to better assess why a *performance-influence model* can be more easily learned as well as identify possible pitfalls that hinder learning. Furthermore, this information might help to improve the proposed machine learning approach, especially if negative influences on learnability are present. As there is a huge variance in software systems, finding systems with similar influences is infeasible and therefore exposing the effects of specific properties of a software system is not unproblematic. To identify the effect of varying some property of a software system on the learnability of a *performance-influence model*, real world software systems are not suitable because we will hardly find another system with only this characteristic varied. Additionally, the problem arises that the

resulting *performance-influence model* can often not be confirmed to be correct as this model is initially unknown for real world data which also complicates finding influences on the learnability.
To solve this issue generated data is required. For this purpose we use an approach proposed by Siegmund et al. [SSA17] that is implemented within the tool *Thor*. This approach allows a realistic generation of input data for *SPL Conqueror* and enables us to vary the data to our preference. The proposed approach attempts to combat the issue of using unrealistically generated influences on configurations of a given software system in the field of research. The main factors of influence on the generated data are the attribute distribution of the influences that can be distributed according to real world systems, the number of interactions between configuration options and the number of participating configuration options in those interactions. Furthermore, the possible configuration options and possible configurations need to be given.

1.1 Goal of this Thesis

Based on this generated data, it might be possible to identify unique influences on the learning of *performance-influence models* and make statements about the general learnability of a software system. Those identified influences might be beneficial for more general assertions when learning a model is hard.
To be able to examine the learnability, there is the need to divide it into certain aspects: therefore, two group of research questions are defined, each containing two questions. Each group corresponds to a specific aspect of possible influences on the learnability.
The first group of questions is concerned with the attribute values of influences on a program configuration. For those values, two relations can be considered: First, the relation between an influence value and the total performance value of a configuration, ergo how important the influence on a configurations is. Secondly, the relation between multiple influences, ergo how their values are distributed over a certain value range, is examined.
This leads to the first set of research questions:

RQ 1.1: How does a change in the relative importance of configuration options affect the learnability?
RQ 1.2: How does a change in the distribution of influences of configuration options affect the learnability?

The second group of research questions focuses on changing the amount and type of interactions among configuration options as well as how those interactions occur. Here, we propose two further research questions: the first question focuses on the amount and order of interaction influences, whereas the second one concentrates on the distribution of those interactions over different orders. The order of an interaction between influences denotes how many influences interact.
This leads us to the following set of research questions:

RQ 2.1: How does the number and order of interactions influence the learnability?
RQ 2.2: How does the distribution of interactions over varying order influence the learnability?

To this end, an experiment design is proposed to examine those aspects of possible influences. Therefore, this thesis will utilize the possibilities uniquely given by *Thor* to evaluate the learnabilty of *performance-influence models* for certain possible influences. The results of the proposed experiment design will be used to answer the research questions.

1.2 Structure of this Thesis

This thesis is structured as follows: First in Chapter 2, the required concepts and tools are introduced. This includes the basic concepts of the tools needed for this thesis, *Thor* and *SPL Conqueror*.
In Chapter 3, the setup of the conducted experiments is explained which includes a description of all independent, dependent and control variables as well as some problems that occurred while conducting the experiments and the influence on the selected variables. According to the previously introduced research questions, the experiments are structured in multiple experiment sets. Subsequently, the changes to *Thor* to conduct the beforehand outlined experiments are presented in Chapter 4. This includes new features as well as multiple corrections of software bugs. The results of the experiments are presented and used to answer the research questions in Chapter 5. In Chapter 6, we discuss possible threats to the validity of our approach and in Chapter 7, an outlook to future work is given. Finally, Chapter 8 concludes this thesis.

2. Background

In this chapter, we provide an overview of the essential basic concepts and required tools. At first in Section 2.1, the process of feature modeling is outlined and an explanation of features and interactions is presented. In Section 2.2 and Section 2.3, an overview of both main programs necessary for this thesis, *SPL Conqueror* and *Thor* respectively, and the underlying concepts is given.

2.1 Feature Modeling

In software product line engineering, feature modeling is an approach to model variability of configurable software systems. All possible configurations of a software system form its configuration space. The purpose is to be able to document all possible elements of the configuration space in an elegant way [ABKS16].

2.1.1 Feature Models

A configuration option can also be denoted as a feature and describes a characteristic of a software system visible to the end-user. The end-user either can, for a binary feature, enable or disable this feature determining whether this characteristic is active for the end-user. A valid combination of features form a *configuration* of that software system [ABKS16].
A feature model is a representation of all possible valid feature combinations and the relationships between those features. The set of all features of the feature model is denoted with $\mathcal{F}$. As the number of possible configurations rises exponentially with the number of features and simply listing those configurations is not feasible, the need for an efficient and elegant representation arises. This can, for example, be achieved by representing the feature model as a tree structure where each node represents a feature. This representation of a feature model is called a feature diagram [ABKS16].
A simple example of a data base system with only the later explained optional features can be seen in Figure 2.1.
As all features can not be freely combined, the following parent-child relations can

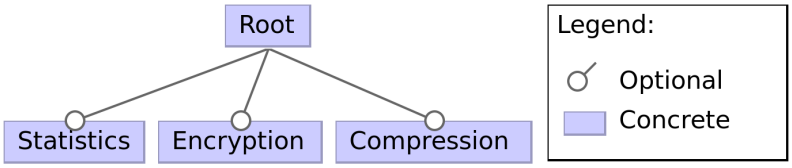

Figure 2.1: Example of a feature diagram whereby the empty bullet denotes an optional feature

be directly modelled in this tree structure. A feature can be *optional* or *mandatory*, meaning that they either can be selected or must be selected if the corresponding parent feature is selected. Furthermore, there are two types of groups: *alternative-groups* and *or-groups*. In an *alternative-group* the selection of the parent feature implies the selection of exactly one child feature. This group corresponds to the logical XOR. The *or-group* corresponds to the logical OR and denotes that at least one child needs to be selected if the corresponding parent feature is selected.
Apart from those parent-child relations, *cross-tree constraints* can exist among features. Those constraints are independent of the tree structure and can be any boolean expression that needs to be fulfilled for a configuration to be valid. Commonly used constraints are *Require* and *exclude* constraints where the selection of a feature either implies or prevents the selection of another feature.
The set of all possible combinations of selected features under those constrains produces the configuration space $\mathcal{C}$ of a software system, whereby each element $c \in \mathcal{C}$ is called a configuration of that software system. A configuration $c \in \mathcal{C}$ can be modelled as a function $c : \mathcal{F} \rightarrow [0, 1]$ that for a feature $f \in \mathcal{F}$ returns if the feature is enabled in this configuration.
This approach serves for a better understanding and analysis of all configurations, but is only capable of modelling functional aspects of configurable software systems. To model other properties like performance or cost, there is the need for a feature model that is extended to additional attributes.

2.1.2 Attributed Feature Models

An attributed feature model is an extension to feature models proposed by Benavides et al. [BTRC05] that incorporates non-functional properties (NFP) into the standard feature model. A NFP is either a continuous value or a discrete value and is normally measured or specified and denotes a performance. This attribute value represents the influence of that feature on the aggregated NFP value of a configuration. This enables additional analyses of a configurable software system like finding the configuration with the least main memory consumption, for example. Nevertheless, as the influence of a feature can differ per configuration where that feature is present, attributed feature models are not enough to fully describe the influence of a feature. This difference in influence comes due to the interaction of features.

2.1.3 Interactions

An interaction occurs among features and influences the NFP of those program configurations where all participating features of that interaction are enabled [Sie12]. Interactions can exist in different orders, meaning that a different amount of features

is involved. In this thesis, we will refer to an interaction of order n if $n+1$ features participate; e.g. an interaction of order 1 means that two features interact with each other.
As observed by Siegmund et al. [SKK+12], interactions in real world systems mainly occur in order 1. Additionally, interactions of order 2 are more likely when a corresponding sub-interactions is present. We denote an interaction as sub-interaction if its participating features are a subset to the higher order interaction. Further observations by Apel et al. [AKS+13] indicate that most interactions are of lower order but also that high order interactions occur.
Interactions can occur in a way that they are easily explained. For example in a database software that has the two features encryption and compression (cf. Figure 2.1), the interaction between those features is easily explained: while each feature negatively impacts the overall performance, selecting both at the same time leads to a proportional better performance as the compressed data is smaller and therefore faster to encrypt. Though, interactions can also occur unforeseen in a way that even the developer can not predict. This may lead to an unexpected drop in program performance.
As we want to consider all possible influences on a configuration, there is the need for a technology to identify those interactions as well as the feature attributes. This can be achieved, for example, with the tool *SPL Conqueror*.

2.2 SPL Conqueror

SPL Conqueror is a tool proposed by Siegmund et al. [SGAK15] to identify influences of features and interactions among them on continuous NFPs in a configurable software system.
As the configuration space can grow exponentially with the number of features, a complete measurement of all configurations and the precedent measurement of the concerned NFP values is often unfeasable. *SPL Conqueror* solves this problem by employing machine learning techniques on a subset of all possible configurations to produce a *performance-influence model* that can be used to predict the NFP values of all configurations.
A *performance-influence model* is essentially a function from the configuration space to a continuous NFP value range: $\Pi : \mathcal{C} \to \mathbb{R}$. This function can always be described as the sum of the minimum base performance, the influences of the individual features and the influences of all present interactions. This model is eminently suitable as it represents all influences on configurations of a software system while having a human readable form.
An example of a *performance-influence model* for the feature model shown in Figure 2.1 with the options *statistics*(S), *encryption*(E) and *compression*(C) can be seen in the following Equation 2.1:

$$\Pi(c) = 100 + 5 \cdot c(S) + 30 \cdot c(E) + 20 \cdot c(C) - 10 \cdot c(E) \cdot c(C) \tag{2.1}$$

SPL Conqueror uses *stepwise linear regression* in combination with *forward feature selection* [CS14] and therefore attempts to fit the *performance-influence model* incrementally. This incremental learning starts with a simple model that considers the influence of no feature. A set of candidate models is generated, each of them

considering another feature of the feature model. Each possible new model is then learned via linear regression and the best model, with regards to the prediction error, is chosen as next candidate. In the next step, further candidate models are generated similarly, again considering one feature of the feature model, but also in combination with the influences existing in the model. This incremental step is repeated until a certain threshold, such as the mean error rate over all configurations and the improvement compared to the previous round, are reached. However, no *backwards elimination* is performed, which would remove unnecessary influences.
SPL Conqueror also implements multiple sampling strategies to enable learning on a subset of the measurement data. Those sampling strategies limit the amount of measurements needed and serve to find a balance between measurement effort and prediction accuracy [SKK+12]. For our purposes, two of the available sampling strategies are utilized: *feature-wise* and *pair-wise* sampling. *Feature-wise* sampling avoids interactions and aims to incorporate a minimal configurations for every feature. Meaning that for every feature, a configurations with that feature enabled and the least amount of other enabled features is selected. This required amount of measurements is therefore linear to the number of features.
Pair-wise sampling aims at identifying interactions of order 1 as this order contains the majority of interactions according to the in Section 2.1.3 mentioned observation. Therefore, for every existing pair of features, the configuration is selected that contains this pair of features and is otherwise minimal.
If an reasonably accurate *performance-influence model* is found it can be used to help developers and end-users likewise. From an end-user point of view, the selection of the desired feature combinations can be simplified. If a certain feature is desired the *performance-influence model* can be used to find all useful as well as all less suitable additional features.
From a developer perspective, as the model explicitly depicts interactions it is straightforward to verify whether a program behaves as expected or if there are any undesired interactions. Additionally, it can be used to gain deeper understanding of the performance of the system as well as identify characteristics that do not fit the expectations of the developer [KSK+18].

2.3 Thor

As current research still relies mostly on attributed feature models with artificial NFP value distributions and mostly ignores interactions, there is the need to realistically generate attributed feature models as well as interactions, where the attribute values follow realistic distributions. *Thor* is a tool proposed by Leutheuss [Leu16] that is able to create attributed feature models with interactions to a given feature model. To this end, the following inputs have to be defined for *Thor*:

- Feature Model
- Number of Interactions
- Interaction Order Distribution
- Feature Attribute Distribution
- Feature Attribute Distribution Scale
- Interaction Attribute Distribution

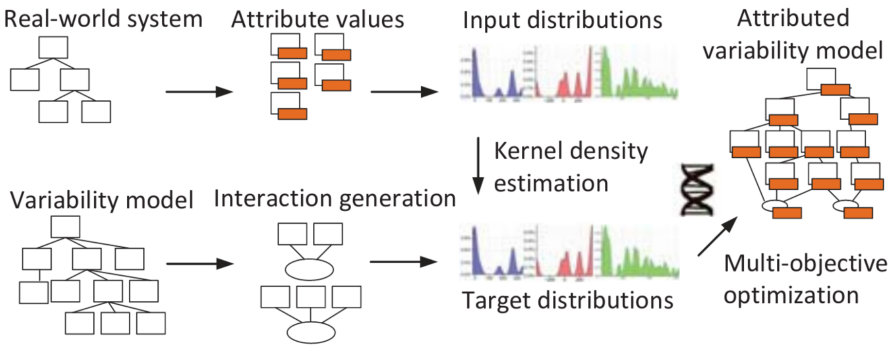

Figure 2.2: Overview of the process of generating attributed feature models [SSA17]

- Interaction Attribute Distribution Scale

The interaction order distribution are float values that represent a percentage of interactions of a specific order. For example, the input 0.5 for order 1 and 0.5 for order 3 means that 50% of the selected interaction number are in order 1 and 50% in order 3. The attribute distributions are lists of float values where each value represents a measured feature or interaction attribute value of a real world system. Those distributions are therefore identified via the name of the software system and the NFP that was measured.
Thor will fit these selected real world distributions to the given feature and interaction numbers which is achieved via kernel density estimation. This is a nonparametric method to estimate the continuous probability measure of a finite dataset. The implementation of *Thor* uses a *Gaussian kernel*. From the estimated probability measure, multiple random samples are taken. The best fitting random sample is then determined via the *two-sample Kolmogorov-Smirnov* test. This test compares two finite samples and checks if those are taken from the same probability measure. The interactions are created randomly, meaning that for an interaction of order 3 *Thor* will select 4 random features and check if those features can exist in a configuration, as is specified by the feature model, as well as if this interaction has not been selected. The resulting feature attribute values as well as the generated interactions and there attribute values compose the output of *Thor*. An overview of this process can be seen in Figure 2.2.
Thor has the additional functionality to generate a subset of all possible configurations where the attribute values of this subset also are similarly distribute to a real world measurement of configurations. This is a multi-objective optimization problem and *Thor* uses a genetic algorithm as a solution approach. This is mainly an issue for feature models with a large configuration space and therefore not relevant for our purpose and not further explained.

2.4 Learnability of Non-Functional Properties

The learnability of the NFP of a configurable software system can be defined as the general quality of the learned *performance-influence model*. The quality of a learned model has two aspects. First, the accuracy of the model to predict the performance

of a configuration and second, the correctness of the model with regards to the existing influences of the input software system.
The first aspect can be described as the mean error rate over all configurations where the results of the model is compared to the actually input configuration value. The second aspect is more complicated as there is the need to syntactically compare the output model to the input influences. This can be done via the tool *PeMoCo* that is explained in Section 2.5.
This is necessary as some influences might only have a small influence on total NFP values of all configurations, thus not effecting the mean error rate. As we use the tool *Thor* to generate those input influences, a syntactical comparison to the output of *SPL Conqueror* to its input is possible.

2.5 PeMoCo

PeMoCo (Performance Model Comparer) is a tool proposed by Kaltenecker [Kal16] that aims at syntactically comparing two *performance-influence models*. Originally, it was intended to compare two performance models of two multigrid programs. Specifically, the aim was to compare the *analytical performance model* created by a domain expert to the *empirical performance model* created by *SPL Conqueror*.
It is utilized to validate if the output *performance-influence model* by *SPL Conqueror* is similar to the via *Thor* defined influences without the need to rely on the error rates of the calculated configuration values.
Generally, the similarity score is computed by assigning each term of the *performance-influence model* a score. The score for a term is in the range $[-2, 2]$ whereby the total similarity score of a model is the sum of all terms of the model.
The score of a term, in a binary setting, can be calculated as follows:

$$score\ of\ a\ term = \begin{cases} -2 & \text{if the other model has no such term} \\ 1 + simValue & \text{if the term exist in the other model with a different value} \\ 2 & \text{if the term exist in the other model with the same value} \end{cases} \tag{2.2}$$

The $simValue$ is defined as $max(0, 1 - e)$ where e is the error rate of the two values of the term. The error rate is defined as $\frac{|x-y|}{y}$ where x is the influence of the term from the via *Thor* generated model and y is the identified model by *SPL Conqueror*.
The possible maximum and minimum score is influenced by the number of existing terms in both models. Therefore, if the number of terms change, the scores are no longer comparable. This implicates that only scores can be compared where the input feature model as well as the number of interactions do not change.
As a result, the utility in some experiments that, for example, rely on changing the number of interactions is reduced. However, it is still generally useful as it enables us to check whether the relevant influences are identified.

3. Experimental Setup

In this chapter, we define the conducted experiments to answer our research questions. To this end, we first present all variables of the experiments – independent, dependent and control.
To focus on the different aspects of the research questions, the experiments are grouped into two sets, each corresponding to two related research questions. For each of those two sets, multiple experiments are performed to examine the influence of different aspects with regards to the related research question.
The first set of experiments addresses the importance of features and interactions and their attribute distribution. With those experiments we aim at answering **RQ 1.1** and **RQ 1.2** as it concentrates on the attribute values of the input.
The second set detains the effects of rising number and different types of interactions and is therefore intend to answer **RQ 2.1** and **RQ 2.2**.
In general, as *Thor* relies on randomness to generate the attribute values and interactions, every experiment was conducted 10 times to account for possible variance in the results. In Section 3.1, we explain the dependent, control and independent variables of the experiments. Additionally, in Section 3.2 some of the problems that were encountered and their influence on the constraints of the conducted experiments are outlined. Finally, in Section 3.3 and Section 3.4, the two sets of experiments are described.

3.1 Variables

In this section, we introduce all variables of the experiments. However, for the independent variables introduced in Section 3.1.1, we use those to only to confine the possible variables we want generally alter. In Section 3.3 and Section 3.4, those variables are further defined as required for the specific experiments.

3.1.1 Dependent Variables

The term dependent variable denotes the outcome of an experiment. This outcome is influenced by the independent variables (see Section 3.1.3) as it depends on those

variables, hence the term.
The output from *SPL Conqueror* is, as mentioned in Section 2.2, a learned *performance-influence model*. As the purpose is to evaluate the learnability of a software system, we limit our dependent variables to the in Section 2.4 defined quality measurements gained from the *performance-influence model*.
The first dependent variable is the computed mean error rate of all configurations of the input feature model. For a single configuration the error rate can be calculated as seen in Equation 3.1. In this equation, *generated* is the performance value of a configuration that is calculated by using the generated feature and interaction attribute values of *Thor* and *predicted* is the value that is computed from the influences defined in the learned *performance-influence model*.

$$error = \frac{|generated - predicted|}{generated} \tag{3.1}$$

The mean error rate allows us to easily compare different results of multiple experiments and is generally applicable.
However, only using the mean error rates brings some drawbacks as it is not able to capture the syntactical difference between the learned and the generated *performance-influence model*, as explained in Section 2.4. To determine these syntactical differences between models the *PeMoCo similarity score* is used which represents the second dependent variable.
These two variables can be effectively used to evaluate the learnability of a *performance-influence model* as both, a semantical and a syntactical result for a learned data set are obtained.

3.1.2 Control Variables

A control variable is one that is not altered throughout the course of the experiment. For our scenario this is foremost *SPL Conqueror* as the tool used to learn the generated data and thus, producing the dependent variables. This tool is so far the only one, to our knowledge, capable to learn a *performance-influence model* on program configurations that are easily human interpretable.
Further dependent variables are the parameters of the machine-learning technique applied in *SPL Conqueror* as well as the parameters of the *Thor* evolutionary algorithm. The variant generation settings of *Thor*, which includes the evolutionary algorithm, are mostly irrelevant as explained in Section 2.3, hence those settings are not changed.
In consideration of the *SPL Conqueror* settings, this thesis focuses on the input data, not the learning settings as the focus is not to find the most suitable settings. This being the case, we use predominantly fixed settings. Furthermore, we always learned on the complete set of measurements, however, data was relearned with the in Section 2.2 explained sampling techniques implemented in *SPL Conqueror* to verify the results as well as to be able to better explain the results.

3.1.3 Independent Variables

In experiments, a variable whose influence on the dependent variables is examined, is called an independent variable.

Since we control the parameters of the machine learning technique used in *SPL Conqueror* and limit our input for *SPL Conqueror* to data generated by *Thor*, its inputs are the main independent variables.
Those inputs, and therefore our independent variables, were previously explained in Section 2.3. Additionally, there is another characteristic of the generated data that we want to consider: The offset of the root feature or the basic influence that every configuration has independent of the selected features or interactions. This offset can not directly be defined in *Thor*. Furthermore, the attribute distributions present in *Thor* do not contain the relation between the root feature attribute value and the feature attribute values. The root feature attribute value is instead equally generated to all feature attribute values and therefore needs to be edited to the desired value previous to learning.
Consequently, each element e of an experiment $\mathcal{E}$ can be described as a tuple $e \in \mathcal{E}$ where
$\mathcal{E} = \{\mathcal{FM} \times \mathcal{IN} \times \mathcal{IOD} \times \mathcal{RO} \times \mathcal{FD} \times \mathcal{FDS} \times \mathcal{ID} \times \mathcal{IDS}\}$ where

- $\mathcal{FM}$ = {Selected Feature Models}
- $\mathcal{IN}$ = {Selected Interaction Numbers}
- $\mathcal{IOD}$ = {Selected Interaction Order Distribution}
- $\mathcal{RO}$ = {Selected Root Feature Offsets}
- $\mathcal{FD}$ = {Selected Feature Attribute Distribution}
- $\mathcal{FDS}$ = {Selected Feature Attribute Distribution Scale}
- $\mathcal{ID}$ = {Selected Interaction Attribute Distribution}
- $\mathcal{IDS}$ = {Selected Interaction Attribute Distribution Scale}

The specific selection of the elements in each of those sets define an experiment series that can be conducted. The scale for feature and interaction is required as the attribute distributions present in *Thor* were measured from different NFPs and therefore the value ranges differ greatly. However, the scale is not necessary when using random functions as attribute distribution because the value range can be influenced by the selected random function itself. A scale s is defined as a tuple of two integers: $s \in \{(n, m) | n, m \in \mathbb{N} \wedge n < m\}$.
In the following sections we describe the conducted experiment series and the constraints under which these defined sets were selected. Beforehand, challenges with early experiments are discussed to better understand some of the constraints that were imposed and their influences on the final experiments.

3.2 Challenges with Initial Experiments

For the first experiment we more extensively relied on the included attribute distributions as well as multiple feature models as we first wanted to detain the effects of varying attribute distributions. The observed results were very random and erratic and consequently, the decision to eliminate as many potential disruptive factors as possible was made. This lead to conducting relative simple base-line experiments where mostly only a single independent variable was manipulated.
One of the factors was the usage of a very simple and basic feature model. Initially, the feature models contained in the *SPL Conqueror* measurements datasets were analysed. This dataset includes NFPs measurements and feature models of

46 software systems, but we restricted the number of considered models to those containing a whole population measurement of all possible system configurations. Based on this analysis, three feature models were selected with varying size in relation to the number of binary features. However, the two larger models were dropped and only the smallest selected model of the software system *BerkeleyC* was used. This is a very simple and basic feature model consisting of only 8 optional binary features and no constraints among those features. This is a valid constraint for our purpose because it merely limits the influence of the feature model which is of no importance to this thesis.
Furthermore, we ascertain that *SPL Conqueror* has difficulties regarding configurations with a NFP value of 0 or below. For all generally known NFPs, negative configurations are not possible. For example, a negative program runtime is impossible. Therefore, this restriction to *SPL Conqueror* is reasonable. However, if we want to utilize negative attribute values, preventing negative configurations is not trivial. Thus, we mostly chose to avoid negative attribute values completely. In Section 4.2 an adjustment to *Thor* is discussed that directly results from this restriction. Besides, for the later needed realistic attribute distributions included in *Thor* the random factor resulting of the limited sample size of some of the distributions has been partially eliminated. As the included attribute distributions consist of a number of float values observed from realistic models, the challenge arises that some included distributions consist only of a small number of float numbers. As the sample size was as low as one or two elements in some occasion, we presume that trying to fit a probability measure onto that sample size leads to varying results and that this contributed to the observed random results. Thus, distributions with less than 10 samples were excluded whereby this is a well-founded decision as it solely concerns a small number of distributions.
Additionally, we detected that *Thor* is unable to generate an individual interaction on its own, ergo, a minimum of two interactions is always generated if interactions are enabled. These constrains lead to the experiment series we define next.

3.3 Attribute Values and Distributions

The experiment series we describe in this section serve the purpose to identify influences on the learnability of *SPL Conqueror* that can be attributed to different feature and interaction attribute values as well as different attribute distributions. Therefore, the independent variables that are focused on are root feature offsets, attribute distributions and scales. The following experiment series is introduced to answer **RQ 1.1** as well as **RQ 1.2** including two experiment series each.

3.3.1 Feature and Interaction Importance

As the influences on the attribute value of a configuration can be split into feature influence and interaction influence, each of those influence is examined separately. In these two resulting experiments we either change the influences of all features or the influence of all interactions on the total attribute value of a configuration. Ergo, vary the importance of features or interactions. These experiments are therefore conducted to answer **RQ 1.1**.
Generally, the importance of a feature or an interaction can be defined as its relation

with the root feature. The root feature is chosen as it is enabled in all configurations and consequently suited to denote the impact of an influence on the NFP value of all configurations. For an influence of the root feature b and an influence value i, the importance relation is defined as the ratio $|\frac{i}{b}| * 100$. Therefore, if the importance of an influence rises, it will have more impact on the NFP value of the configurations where it is enabled.
For our case however, we will consider this importance relation only for all features or interactions collectively. This is done to simplify the required experiment design as well as to allow for better analysis of the results. To this end, normal distributions are used to generate all attribute values as the center of attention is exclusively the value range of the attributes, but not the distribution of those attributes. The importance relation can therefore be given as the relation not between a single influence but between the mean of the selected normal distribution and the root offset.
For both importance experiments, the value of the root feature offset will be set to 1000 as this allows for a reasonable amount of percentages to be utilized without generating values that are close to zero. Furthermore, the value of the standard deviation of the normal distribution is set to 10, to actually generate a range of elements that is not to broad. Additionally, no scales are needed as the value range is modified via the mean of the attribute distribution.

Experiment 1: Feature Importance

To consider relevant importance values, the selected percentages are mainly small. Therefore, the selected percentage values are {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 25, 50}. Furthermore, no interactions were chosen to also eliminate this possible influence.
As the percentage values are very low, the possibility exists that negative values are generated due to the selected standard deviation. However, this should only occur for experiments where 1 % or possibly 2 % are chosen as importance value. Additionally, we do not expect this to be a large influence. In general, this leads to the following experiment series:

- $\mathcal{FM} = \{\text{BerkeleyC}\}$
- $\mathcal{IN} = \{\emptyset\}$
- $\mathcal{IOD} = \{\emptyset\}$
- $\mathcal{RO} = \{1000\}$
- $\mathcal{FD} = \{\mathcal{N}(\mu, 10) \mid \mu \in \{r \cdot x\% \mid x \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 25, 50\} \wedge r \in \mathcal{RO}\}\}$
- $\mathcal{FDS} = \{\emptyset\}$
- $\mathcal{ID} = \{\emptyset\}$
- $\mathcal{IDS} = \{\emptyset\}$

Experiment 2: Interaction Importance

Again, only considering interesting importance percentages, the selected percentage values are {1, 5, 10, 15, 25, 50, 75, 100, 150, 200, 250, 500, 1000}. For this experiment we select two interactions that are both of order 1. This is sufficient as only the influence of interactions is the center of attention whereas the influence of multiple interactions as well as order is a concern in the experiments in Section 3.4.

Additionally, possible influences that arise due to too many interactions are avoided. Again for the 1% importance value, negative interactions are possible. This leads to the following sets defining this experiment series:

- $\mathcal{FM} = \{\text{BerkeleyC}\}$
- $\mathcal{IN} = \{\frac{1}{4} \cdot n \mid n = \text{no. Features}\}$
- $\mathcal{IOD} = \{Order\ 1\}$
- $\mathcal{RO} = \{1000\}$
- $\mathcal{FD} = \{\mathcal{N}(r,\ 10) \mid r \in \mathcal{RO}\}$
- $\mathcal{FDS} = \{\emptyset\}$
- $\mathcal{ID} = \{\mathcal{N}(\mu,\ 10) \mid \mu \in \{r \cdot x\% \mid x \in \{1,\ 5,\ 10,\ 15,\ 25,\ 50,\ 75,\ 100,\ 150,\ 200,\ 250,\ 500,\ 1000\} \wedge r \in \mathcal{RO}\}\}$
- $\mathcal{IDS} = \{\emptyset\}$

3.3.2 Feature and Interaction Distributions

To answer **RQ 1.2**, we perform two experiment series with the focus on either features or interactions.
For these experiments, the main focus are the attribute distributions included in *Thor*. As mentioned in Section 3.2, there is a restriction which distributions are actually used.
As the distributions are taken from multiple real world NFPs, there is a huge difference in value range between these distributions. Therefore, the distributions need to be scaled to be able to compare them and to solely consider the influence of the attribute distributions without an influence of the value range. The scale minimum is set to 1 to avoid generating values that are very low. In some of the included distributions, there exist an accumulation of minimal values and this accumulation then occurs at the scale minimum. Originally, the minimum was planed at 0, but this leads to very low values with little influence and therefore no relevance. The scale maximum is varied for both experiments as the value range between scale maximum and scale minimum could also influence the results. To this end, the selected values for the scale maximum are $\{10, 50, 100, 500, 1000, 5000, 10000\}$. Increasing the scale maximum leads to an increase of the generated attribute values. If the root feature remains constant, this consequently increases the relative importance of all features. As this is another influence that we want to avoid, the root feature must also be varied. Therefore, the root feature offset is always set to the maximum of the selected scale to further limit the influence of varying attribute importance. Thus, the relative importance remains the same for both feature and interaction attribute values even if the scaling changes.

Experiment 1: Feature Attribute Distribution

Regarding the experiments with focus on the attribute distributions of features, interactions were ignored as only the influence of changing the distributions is relevant. These requirements lead to the following total experiment series:

- $\mathcal{FM} = \{\text{BerkeleyC}\}$
- $\mathcal{IN} = \{\emptyset\}$

- $\mathcal{IOD} = \{\emptyset\}$
- $\mathcal{RO} = \{m\}$
- $\mathcal{FD} = \{x \in \{\text{Feature Attribute Distributions of Thor}\} \mid |x| \geq 10\}$
- $\mathcal{FDS} = \{(n, m) | n = 1 \wedge m \in \{10, 50, 100, 500, 1000, 5000, 10000\}\}$
- $\mathcal{ID} = \{\emptyset\}$
- $\mathcal{IDS} = \{\emptyset\}$

Experiment 2: Interaction Attribute Distribution

Considering the experiments varying the interaction attribute distribution, the number of interaction was set to 8. This is done not only to have a number of interactions that can actually be differently distributed but also to match the number of features in our selected feature model.
In this experiment, we do not vary the order of the interactions but only set it order 1 as those interactions are more realistic as mentioned in Section 2.1.3. Furthermore, the effect of varying the order of interactions is examined with the experiments in Section 3.4. Apart from setting the root feature offset to the current scale maximum, the feature attribute distribution is also set to $\mathcal{N}(m,\ 1)$ for the currently selected scale (n, m) as the importance of a feature should remain equal for the course of the experiment and therefore does not effect the results.

- $\mathcal{FM} = \{\text{BerkeleyC}\}$
- $\mathcal{IN} = \{n \mid n = \text{no. Features}\}$
- $\mathcal{IOD} = \{\text{Order 1}\}$
- $\mathcal{RO} = \{m\}$
- $\mathcal{FD} = \{\mathcal{N}(m,\ 1)\}$
- $\mathcal{FDS} = \{\emptyset\}$
- $\mathcal{ID} = \{x \in \{\text{Interaction Attribute Distributions of Thor}\} \mid |x| \geq 10\}$
- $\mathcal{IDS} = \{(n, m) | n = 1 \wedge m \in \{10, 50, 100, 500, 1000, 5000, 10000\}\}$

3.4 Interaction Number and Order Distribution

To answer **RQ 2.1** and **RQ 2.2**, we consider the influence of the number of interactions, the order of interactions and order distribution of interactions on the learnability. To this end, we define two experiment series with this focus: the first for varying interaction number and order with regards to **RQ 2.1** and the second according to **RQ 2.2** regarding varying order distributions of interactions.

3.4.1 Number of Interactions

The first experiment series focuses on the number of interactions. Thereto, the maximum number of possible interactions of a specific order has to be known which is given by the feature model as well as the specific order of the interactions. Generally, the maximum number of interactions for a single order n is every combination of $n + 1$ features from the feature model under the restriction that this feature combination exists in a valid configuration. This is not a concern as the used *BerkeleyC* feature model consist solely of optional features and thus, every combination of n features exists in a valid configuration. Therefore, in our case the maximum number of interactions for the order n is the binomial coefficient $\binom{8}{n+1}$. All maximum

Order	1	2	3	4	5	6	7
Maximum number of interactions	28	56	70	56	28	8	1

Table 3.1: Maximum number of interactions per order

interaction numbers can be seen in Table 3.1.

As mentioned in Section 3.2, *Thor* can not handle generating only a single interaction and therefore the sole interaction of order 7 is ignored. Apart from that, we wanted to include a high number of interactions as well as all other orders. Thus, the order of an experiment defines the possible maximum number of interactions. This leads us to the selected interaction numbers. We wanted to have a reasonable range of numbers with a reasonable extent. This leads to the following set of interaction numbers: $\{2, 3, 4, 5, 7, 10, 15, 20, 25, 35, 50\}$.
We refrained from using the highest possible number, 70, to be able to compare the highest selected number over multiple orders.
Consequently, the following experiments series is specified:

- $\mathcal{FM}$ = {BerkeleyC}
- $\mathcal{IN}$ = $\{n \mid n \in \{2, 3, 4, 5, 7, 10, 15, 20, 25, 35, 50\} \wedge n <$ max no. of interactions in current order$\}$
- $\mathcal{IOD}$ = {Order 1, Order 2, Order 3,Order 4, Order 5, Order 6}
- $\mathcal{RO}$ = $\{\emptyset\}$
- $\mathcal{FD}$ = $\{\mathcal{N}(25, 10)\}$
- $\mathcal{FDS}$ = $\{\emptyset\}$
- $\mathcal{ID}$ = $\{\mathcal{N}(100, 10)\}$
- $\mathcal{IDS}$ = $\{\emptyset\}$

3.4.2 Interaction Order Distribution

Finally, the experiment series which focuses on the distribution of interactions over different orders are defined to answer **RQ 2.2**. In this experiment, we do not change the number of interactions but only the distribution of those interactions among varying order as we want to compare the different distributions.
We also aim at considering a large number of interactions to be able to effectively distribute interactions onto different orders. As seen in Table 3.1, 28 interactions is the highest value that a majority of orders can support. Therefore, only orders up to the maximum of 5 are present while orders 6 and 7 can not have that many interactions, hence will be ignored. For reasons of simplification, a number of 25 interactions was chosen as it was easy to effectively distribute uniform over 5 different orders.
To define a distribution of interactions among orders, we can give the concrete numbers of interactions per order. This is done as both number of interactions as well as the orders are fixed and finite. To avoid further factors, only simple distributions are incorporated. Additionally, distributions were not exclusively selected according to real world interaction order distributions.
The uniform distribution of the interactions over all orders was selected as a base line

result. Additionally, a more realistic interaction order distribution where the number of interactions is descending as the order is increasing is included. To introduce the opposite distribution, its inverse is included whereby an ascending number of interactions is distributed to higher order. Eventually, the interactions are normal distributed over the orders to also include a distribution where the maximum number of interactions is not present in order 1 or 5. Once more an inverse of this distribution is included, with a distribution where the number of interactions is descending from order 1 to order 3 and afterwards is ascending up to order 5. This forms a kind of inverse normal distribution. This leads to the following distributions and concrete interaction numbers seen in Table 3.2.

Eventually, to eliminate additional factors, the attribute distributions for both

Distribution	Order 1	Order 2	Order 3	Order 4	Order 5
Uniform	5	5	5	5	5
Descending	9	7	5	3	1
Ascending	1	3	5	7	9
Normal	3	5	9	5	3
Inverse Normal	8	4	1	4	8

Table 3.2: Selected discrete order distributions of interactions

features and interactions are set to a normal distribution with a standard deviation of 10. According to the previous defined experiments with regards to the interaction number, the mean of the attribute distributions of interactions is set to 100. Again, this is done as *SPL Conqueror* should be able to detect the interactions and therefore the observed results are due to the different distributions, not due to the undiscovered interactions. This also makes the results comparable to those of the interaction number experiments considering only a number of 25 interactions and can therefore also see the effect of 25 interactions in one order. Overall, this leads us to the following experiment series:

- $\mathcal{FM}$ = {BerkeleyC}
- $\mathcal{IN}$ = {25}
- $\mathcal{IOD}$ = {Uniform, Ascending, Descending, Normal, Invers-Normal}
- $\mathcal{RO}$ = {$\emptyset$}
- $\mathcal{FD}$ = {$\mathcal{N}$(25, 10)}
- $\mathcal{FDS}$ = {$\emptyset$}
- $\mathcal{ID}$ = {$\mathcal{N}$(100, 10)}
- $\mathcal{IDS}$ = {$\emptyset$}

4. Implementation

In this chapter, we describe the changes and novel features that were implemented in *Thor* to enable conducting the previously outlined experiments as well as the auxiliary.
In general, *Thor* as well as *SPL Conqueror* were executed on a computer with an `Intel Core i5-4590 @ 3.30 GHz` with 4 CPU cores and `16 GB RAM`. To this end, the generation of the required input for *Thor* as well as the execution of *Thor* was automated by Python scripts. This allowed us to start generating an experiment series as a total and afterwards simply starting *SPL Conqueror* with each single dataset. All scripts written for this purpose are available on GitHub[1].
Apart from the Python scripts, multiple additions and corrections of software bugs were required for *Thor* to enable the experiments and to minimize the number of random factors.
In Section 4.1, the changes necessary to enable the usage of random functions and in Section 4.2 the changes to enable scaling the attribute distributions are explained. Any further additions and corrections are illustrated in Section 4.3 before in Section 4.4 some challenges in working with *Thor* are outlined.

4.1 Random Functions

As mentioned in Section 3.2, there is the need to eliminate as many factors as possible and therefore, random functions were used as attribute distributions. *Thor* implements two random functions: uniform distribution or normal distribution respectively. Despite this, the implementation in *Thor* was unsuitable for our purpose because the results were too random as *Thor* only took a single random sample from the selected random function.
We implemented an additional function (see Listing 4.1) that takes multiple samples from the selected random function and returns the best fitting sample. The best fitting sample is determined using the *one sample Kolmogorov-Smirnov* test. This test compares a sample to a probability measure and estimates if this sample was

[1] https://github.com/JakobHoffmann/GenerateSPLConquerorExperiments

drawn from that random distribution. We applied this methodology based on the pre-existing implementation in *Thor* to scale the selected attribute distributions to the given problem sizes, e.g. the number of features. *Thor* estimates the probability measure of the selected attribute distribution via kernel density estimation and then takes multiple random samples of the given problem size. This sample is compared to the original measurement data and the best fitting one is selected. To this end, *Thor* uses a *two sample Kolmogorov-Smirnov* test. As we do not have to estimate a probability measure but have it given as our input, we can resort to using the *one sample Kolmogorov-Smirnov* test.
Our implementation can be easily used for more random functions than the two currently accessible in *Thor*. This is also utilized as we added the possibility to select an exponential random function.

```
private double[] getDistributionSample(int amount, int rounds,
                UnivariateContinuousDistribution dist)
{
  double[] values = new double[amount];
  double PValue = 0.0;
  for (int i = 0; i < rounds; i++)
  {
    double[] temp_values = dist.Generate(amount);
    KolmogorovSmirnovTest test =
      new KolmogorovSmirnovTest(temp_values, dist);
    if (PValue < test.PValue)
    {
      values = temp_values;
      PValue = test.PValue;
    }
  }
  return values;
}
```

Listing 4.1: Function `getDistributionSample`

4.2 Scaling Attribute Distributions

To utilize the included realistic attribute distributions included in *Thor*, we need to scale those to a common value range as explained in Section 3.3.2. *Thor* has already such scaling method, but it is implemented in such a manner that the minimum and maximum value of all attribute values would always be selected. In case of scaling the attribute values between 0 and 10, a feature with the attribute value 0 and one with 10 would always exist. This should be avoided as it defeats the purpose of generating realistic data. This scaling method is applied after the generation of the attribute values.
Originally, we thought that applying this exact scaling to the input distribution values would solve this problem as only the input would contain the fixed values while the generated data differs. However, *Thor* uses kernel density estimation to fit a probability measure to the input distribution and samples from this measure. Thus, negative attribute values and therefore negative configurations are not avoided even if the original input contains no negative values. This can be clearly observed in Figure 4.1 where the histogram and estimated probability measure of the in *Thor*

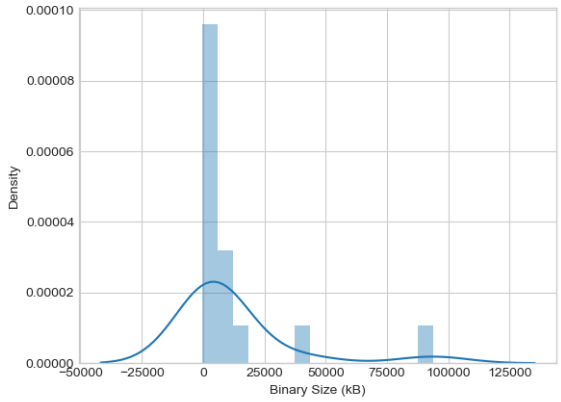

Figure 4.1: Density plot of the estimated probability measure for the *BDB BinarySize* sample

contained *BDB BinarySize* interaction attribute sample is shown. For our purpose, negative attribute values need to be prevented as negative configurations need to be avoided (cf. Section 3.2) and implementing a feasible verification if all configurations are positive, is not within the scope of this thesis.
Generally, negative attribute values are explicitly desirable in *Thor* as they occur in real world models and consequently there was no function implemented to avoid those. Additionally, as *Thor* assigns the generated attribute values at random, there is no mechanism in place to avoid negative configurations.
As mentioned in Section 3.2, *SPL Conqueror* is unable to manage those negative configurations which is according to real world expectations. Continuous NFPs values are always consumption of resources and therefore, the total value of a configuration is positive. For example, a configuration with negative main memory usage implies that executing the program with that configuration increases the main memory of the system.
Increasing the minimum of the scale is no problem-solving approach as increasing the maximum of the scale is also desired. This is due to the effect that if the maximum of the scale is increased enough to reach the previous ratio between minimum and maximum, the random sample from the estimated probability measure will again contain negative attribute values.

The solution approach is to edit the method how *Thor* takes a random sample from the estimated probability measure. As the distributions are scaled beforehand, we expected to only have some negative attribute values. Therefore, we doubled the sample size and removed all values outside the bounds given by the selected scale. Ensuing, if the remaining values are bigger than the needed sample size, values are chosen randomly until the needed sample size is reached. This change takes effect before the *Kolmogorov-Smirnov* goodness of fit test is applied, meaning that this procedure is executed multiple times and solely the final best candidate is chosen. This is the reason why this approach does not distort the result as we do only avoid out of bound values, but do not prohibit the original procedure to work as intended. This implemented functionality enables us to scale the attribute values over different ranges and still avoiding unrealistic given values. Apart from the explained func-

tionality additions to *Thor*, further improvements and additions are explained in the following subsection.

4.3 Improvements to Thor

Some software bugs as well as the need of additional enhancements were identified while using *Thor*.
Regarding the command-line, *Thor* was not terminating after completing the data generation. As the data is written by another thread, this was done intentionally to ensure that this thread is not aborted prematurely. We solved this issue as it was essential to automate our data generation as a huge number of experiments is generated.
The command-line in *Thor* would not allow selecting random functions. Therefore, commands are added for the existing random functions as well as the newly added exponential random function. However, this functionality is enabled for the existing random functions in the *Thor* graphical user interface (GUI).
Additionally, the commands for scaling the attribute distributions were not implemented in the command-line. This functionality is again enabled in the GUI and commands to access it from the command line were added. This is done independently of the modification to the scaling in *Thor* mentioned in Section 4.2.
As all of those commands were already implemented in the GUI of *Thor* this leads to the conclusion that the implementation was probably planned, but subordinated to other features.
Furthermore, two software bugs while generating interactions in *Thor* that also needed to be corrected were identified. The method for checking whether an interaction already exist was responsible for the first error as it would falsely identify an interaction as existing if a sub-interaction of that interaction exists.
Additionally, the method iterating through all orders and creating the required interactions was flawed. For every increase of order, an interaction was created, i.e. the validation if an interaction in the current order was required was only done after creating the first interaction. The integration of a loop that increases the interaction order until an interaction actually needs to be created, solved this issue.
A smaller software bug occurring while loading an attribute distribution due to not parsing the input correctly, was also rectified. We also added another command to pass a random seed that enables reproducibility.
Regarding the documentation, a pull request was made for every additional implementation was made as well as a issue for each found error was opened in the *Thor* GitHub repository[2].

4.4 Challenges with Thor

As previously mentioned in Section 4.3, we identified and rectified some software bugs in *Thor*. Besides, some additional challenges have arisen while using *Thor*, as the for our purposes indispensable command line was probably just an afterthought. Therefore, the following errors are specific to the command line and may not be

[2]https://github.com/se-passau/thor-avm

present while using the GUI. Furthermore, no attempt was made to exactly identify and or solve these errors as it was out of the scope for this thesis.
Due to missing input checks in *Thor*, some errors can be attributed to false input. Additionally, some crashes were not due to input errors but rather to additional software bugs. For example, *Thor* can not generate a sole interaction which should typically be possible, but it simply seems to be some counting error in a loop. Furthermore, *Thor* also crashes if there is no attribute distribution given to *Thor* for either interactions or variants, even with those functions are set to false. It seems that the flag for both variant and interaction calculation has no effect whereas in the GUI disabling this functionality via those flags is possible.
Overall, the combination of software bugs, missing defensive programming and false input made identifying those bugs very time consuming and tedious. In hindsight, we only needed a small subset of the functionality of *Thor*, reimplementing those functions could have been a viable option. As a direct consequence, the presence of these errors and pitfalls lead to a reduction of the feasible experiments, as the working time on this thesis is limited.

5. Evaluation

In this chapter, the results of the conducted experiments are presented and the conclusions to the research questions are discussed.
In general, as mentioned in Section 3.1.2, we also repeated all experiments with the *feature-wise* and *pair-wise* sampling strategy. Those results are also discussed, however, we do not evaluate them to the full extent, but use them to verify our results.
To visualize the resulting data, we resorted to using box-plots as those are most suitable for the requirement of depicting a certain value range for a specific result. Violin-plots could have also be used as they generally can give more information about the specific distribution of the data. However, for the following experiments, this was never the case as most of the results had not enough data and we further saw no benefit but rather a reduced visibility of the results.

5.1 Effects of Attribute Values and Distribution

First, according to the separation of the experiments with regards to **RQ 1.1** and **RQ 1.2**, the results for the importance and the value distribution experiments are presented.

5.1.1 Feature and Interaction Importance

According to **RQ 1.1**, we use the following two experiments to identify influences on the learnability that result from varying importance of the influences. The following figures show two plots, the error rate in percent on the y-axis of the left side plot and the *PeMoCo* score on the y-axis of the right side plot. The x-axis displays the in Section 3.3.1 defined importance percentage of the considered influence and is the same for both plots. The shown data points are the results from the 10 repetitions for each importance value and are depicted as a box plot.

Experiment 1: Feature Importance

In Figure 5.1, it can be observed that for feature importance values higher than 25 %, the error rate as well as the *PeMoCo* score indicates a perfect result as the

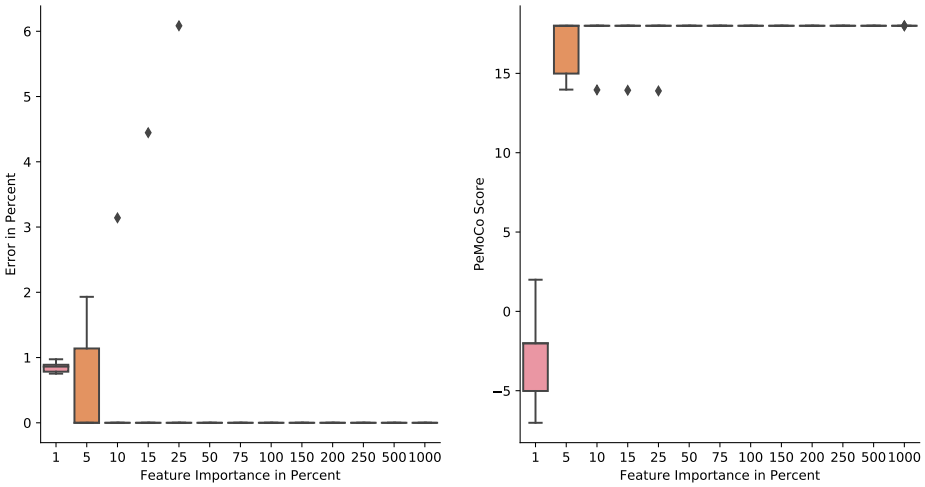

Figure 5.1: Error rate and *PeMoCo* score of the feature importance experiment showing higher percentages

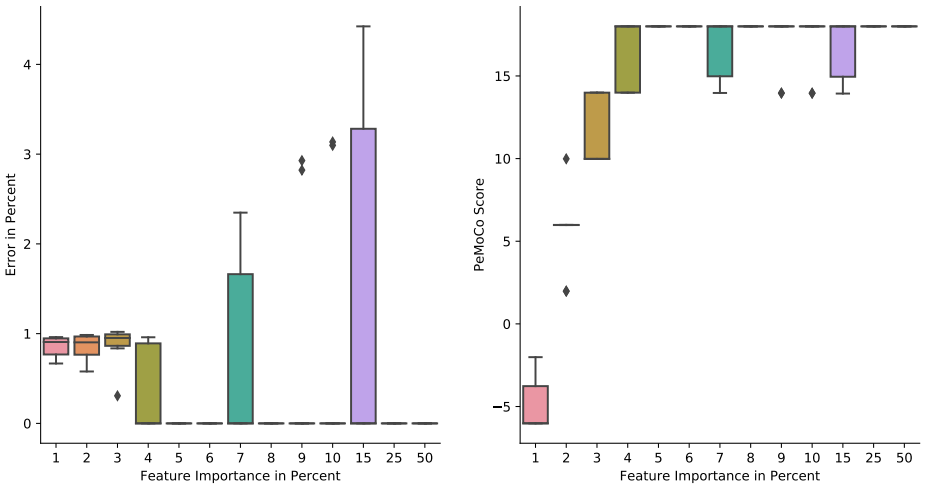

Figure 5.2: Error rate and *PeMoCo* score of the feature importance experiment

error is minimal and the *PeMoCo* score is maximal. Therefore, lower importance values were selected for the experiment with regards to feature importance, seen in Figure 5.2, where those perfect results only occur for the 25 % and 50 % importance. For lower importance percentages, seen in Figure 5.2, there are a few experiments with a higher error rate. Additionally, for the lowest numbers of importance percentages from 1 % to 4 %, the error rate does no longer have a large standard deviation and is always around 1 %. For those results, *SPL Conqueror* is no longer able to identify all influences whereas for all higher results, a perfect model is most of the times identified.
Considering the *PeMoCo* score, a clear decrease is observed for those low importance percentages. However, this does not lead to an increase in error rate for those low importance numbers. It is also notable that the results for the importance values 4 %, 7 %, 9 %, 10 % and 15 % show that the observed error is always due to a single unidentified feature as the score indicates. Furthermore, if an error is present, it increases with increasing importance which is according to the definition of importance in our context. The importance denotes a higher influence on the NFP of a configuration and thus, the error rate. The lower importance can also explain the absent rise in error rate for the lower importance percentages, that can be expected due to the lower *PeMoCo* score. However, as the influence of those unidentified features is negligible, the increase in unidentified features is compensated. Thus, the error remains approximately the same.
This leads to the conclusion, that as the features are less important, the probability to identify those decreases. This explains the erratic results for the importance values from 4 % to 15 % where only single results have a higher error rate and most models are perfectly identified. Those can be attributed to random factor of the data generation, however, we see an overall trend. There is a point in importance, around 15 %, where the probability that an feature is not identified is no longer very small, and this probability increases while the importance decreases. Furthermore, regarding the importance percentage of 1 %, the *PeMoCo* score indicates that only a remaining small number of features is identified. Decreasing the importance further should lead to a further decrease in *PeMoCo* score. Once the minimum *PeMoCo* score is reached, the error rate should also decrease as the influence of the options become less relevant.
Finally, a skewed peak in error rate can be observed for the low importance numbers smaller than 15 % as the number of found influences decreases. Once the importance is so small that no influences are found, the error will drop again as the importance decreases further.

Varying the sampling does show the same observed results. However, there are two interesting characteristics: for the *feature-wise* sampling seen in Figure 5.3 and *pair-wise* sampling seen in Figure 5.4, the possible error rate is higher. Additional analysis of the data showed that the improvement per round reached the required minimum and this improvement is measured on the sampled data. Regarding the *feature-wise* sampling as well as for the *pair-wise* sampling, the training set gets reduced in a way that distorts the amount of occurrences of influences. Meaning, that in the training set an influence will effect a smaller number of configurations than before. As the configurations space of the *BerkeleyC* feature model is the power set of the set of features, a single feature is present in half of the configurations. Thus,

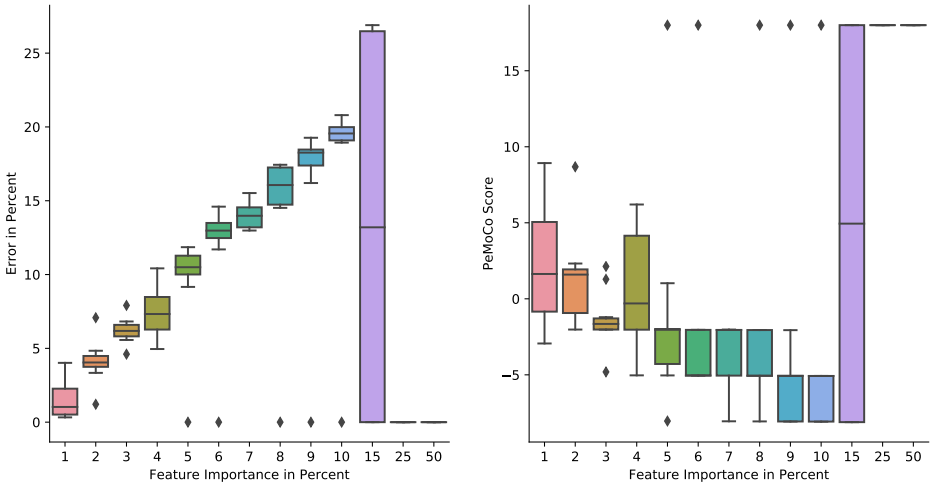

Figure 5.3: Error rate and *PeMoCo* score of the feature importance experiment with the *feature-wise* sampling strategy

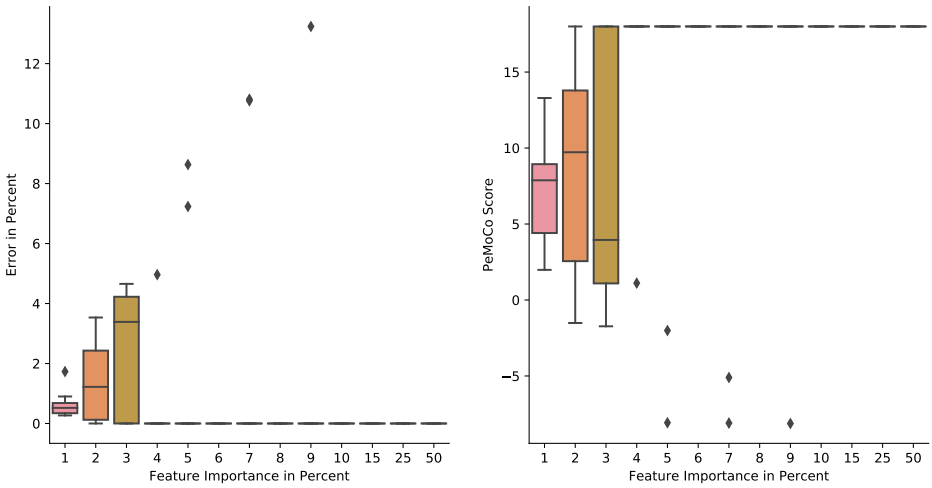

Figure 5.4: Error rate and *PeMoCo* score of the feature importance experiment with the *pair-wise* sampling strategy

a single feature is present in 50 % of all configurations whereas for the *feature-wise* sampling as for every feature a minimal configurations is selected, it is only present in 12.5 % of all configurations. This leads to the higher error rate as the error rate is only optimized to this reduced training set and the effect of an unidentified feature is higher while regarding the error rate on all configurations. We expect this effect to be also present for *pair-wise* sampling as the maximum error rate is also higher compared to the experiment while using the complete data for learning.
The second characteristic can again be observed for the *pair-wise* sampling (cf. Figure 5.4) as well as for the *feature-wise* sampling (cf. Figure 5.3). In contrast to the experiment regarding the feature importance, the *PeMoCo* score does not increase for lower importance values. On the contrary, the lowest *PeMoCo* scores can be observed for the highest importance values with an error rate different than zero. Again, analysis of the learned *performance-influence models* showed that the number of present features in a model remains approximately the same for higher importance values, however, the values of the identified features are incorrect. For the *feature-wise* sampling as *SPL Conqueror* only learns on configurations where only a single feature is enabled and as all features have a similar attribute value, it is sufficient for the error rate to just assume a higher root feature value. This problem arises as the via the *feature-wise* sampling reduced training set is not representative of the whole configuration space. For example, considering the 10 % importance value and while only testing on configurations with only a single feature enabled, the error rate is low if *SPL Conqueror* just learns a root feature value of 1100. Therefore, the remaining identified features all have a value around the standard deviation of 10. For an importance value of 1 %, the root feature value is approximated to 1010 whereas the identified features have the same attribute value around the standard deviation of 10. As the influence of the features are therefore more correct, resulting in the observed increase of the *PeMoCo* score for lower importance values. We expect this observation to also hold for *pair-wise* sampling in a similar manner.

Experiment 2: Interaction Importance

Regarding the experiments where we change the importance of the interactions, seen in Figure 5.5, the same skewed peak already observed for the feature importance experiment exists. As this experiment was conducted with only 2 interactions, the *PeMoCo* scores, apart from the experiment where the importance is set to 200 %, are always around three values. A *PeMoCo* score of approximately 14 represents no identified interaction whereas the scores 18 and 22 represent 1 and 2 identified interactions respectively. This aggregation also implies that the features are always identified. The result for a 200 % importance value, has the curiosity that there is always an interaction identified with a very small value between e^{-12} and e^{-13}. This has no influence on the error rate, however, it reduces the *PeMoCo* score. We were not able to explain this characteristic, but it is not relevant for the general tendency and can therefore be ignored.
Compared to the feature importance results, the importance percentage needed that *SPL Conqueror* identifies an interaction is much higher. For an importance percentage of 250 or higher, all interactions are found, resulting in a perfect model. The highest error rate is again reached for the highest importance where no longer all interactions are found. The number of found interactions drops from 25 to 15 % and

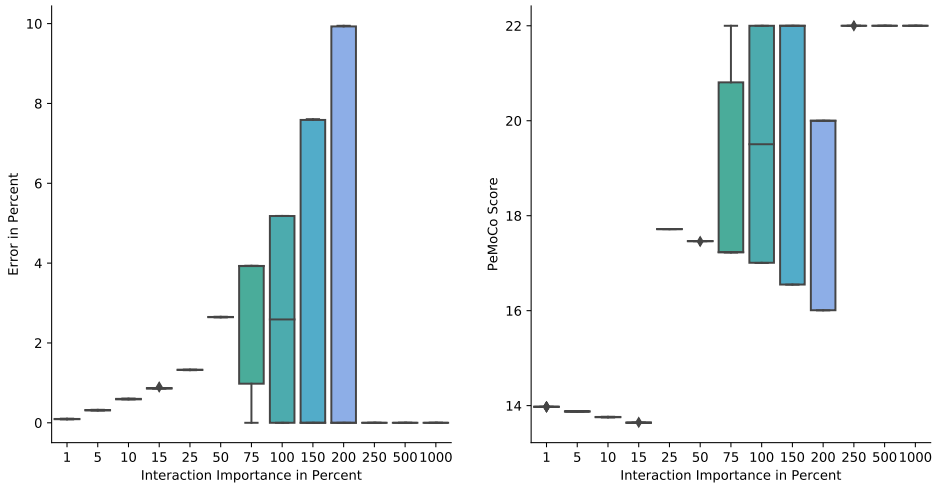

Figure 5.5: Error rate and *PeMoCo* score of the interaction importance experiment

is not reflected in the error rate as their influence is small. The continuous drop in error rate, which we predicted for the feature importance experiment, can be also observed for this experiment.

The results for the *feature-wise* sampling technique is ignored as no interactions are found and the resulting plot can be found in the Appendix. Considering the results with *pair-wise* sampling, seen in Figure 5.6, a larger variance can be observed. The results for importance values higher than 100 % can probably be attributed to unidentified feature influences as the *PeMoCo* score indicates. The observation for unidentified interactions, illustrated in Figure 5.5, is also present, but for much lower importance values. For an importance value of 10 % and below, the *PeMoCo* score drops and the error increases slightly. This indicates that the importance value for the *pair-wise* sampling needs to be much lower for an interaction to be no longer identified. The high error rate for the importance values above 100 %, however, is not explained as the previous observations are still present and this seems to be due to the *pair-wise* sampling.

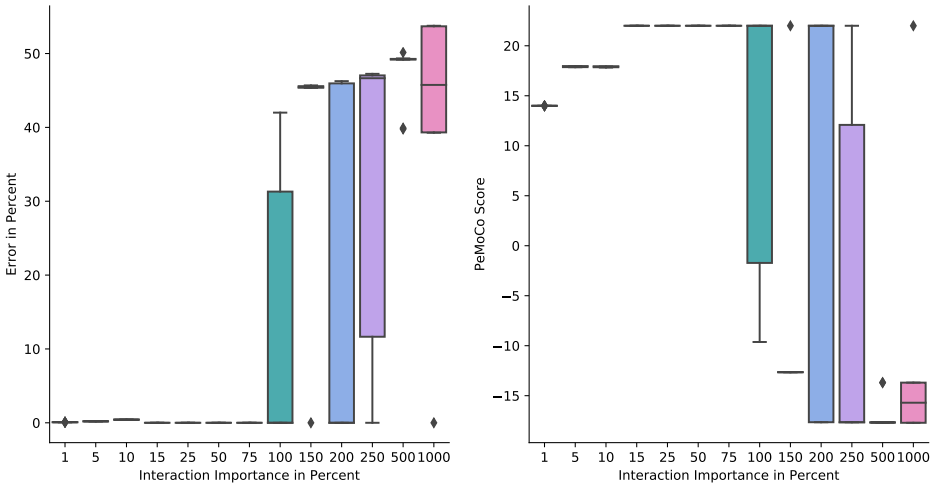

Figure 5.6: Error rate and *PeMoCo* score of the interaction importance experiment with the *pair-wise* sampling strategy

5.1.2 Feature and Interaction Distribution

With the following two experiments, we want to identify influences on the learnability that result from varying attribute distribution of the influences. This corresponds to **RQ 1.2**. Furthermore, the distributions were scaled to varying maximums.
The following figures show two plots each, with the error rate and *PeMoCo* score being the respective x-axis whereas the varying attribute distributions or the selected scale maximum can be seen on the y-axis. If not further specified, while considering the attribute distributions, the shown data points are the results from the 10 repetitions for each scale and vice versa. For each of the two experiments, there is an additional figure that consist only of one plot and shows the generated attribute values on y-axis whereas the x-axis shows the attribute distributions.

Experiment 1: Feature Attribute Distribution

Apart from the outliers, the feature attribute distribution results are overall very similar and the error rate is mostly below 1 % (cf. Figure 5.7). However, it can be seen in the *PeMoCo* result that the increasing error mostly corresponds to an decreasing *PeMoCo* score. Therefore, the error can be attributed to influences that were not identified.
To evaluate these results, we additionally need to consider the generated values. In Figure 5.8 those values for the maximum scale of 100 are illustrated whereby it must be noted that the relation between those values does not change significantly for different scales. The plots for the other scales can be found in the Appendix. This plot demonstrates that the results with highest error rate have a similar generated value distribution whereby most values are near the scale minimum, especially the *SNW BinarySize* and *Violet BinarySize* attribute distributions, with only a few outliers (cf. Figure 5.8). This indicates, that those values have a lower relative importance compared to the root feature value as it always is the scale maximum. Additionally,

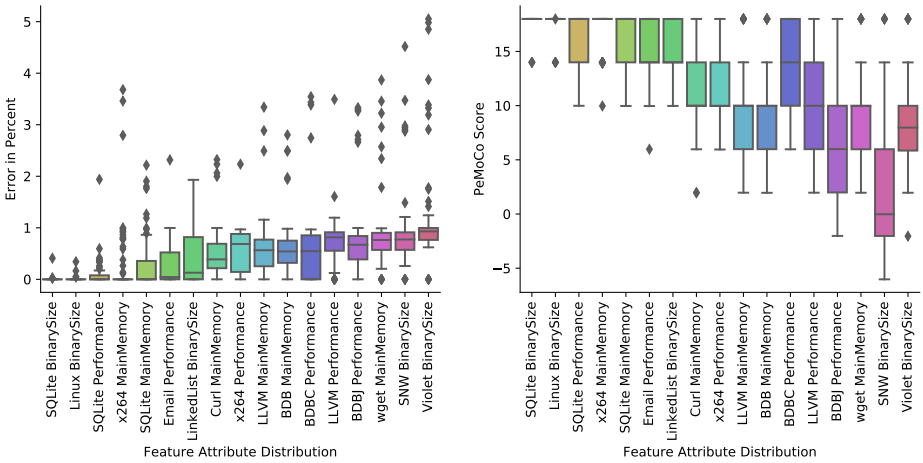

Figure 5.7: Error rate and *PeMoCo* score of the feature attribute distribution experiment considering only the attribute distributions independent of the scale

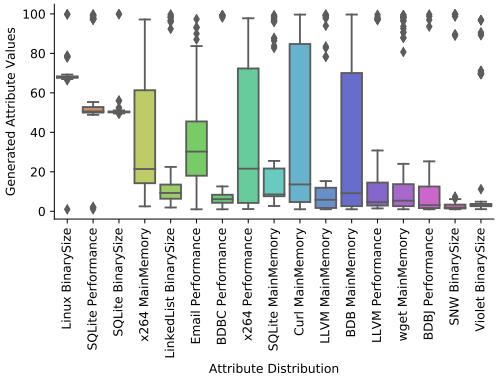

Figure 5.8: Generated feature attribute values for a scale maximum of 100

for the results with the smallest error rate, there is also a similarity in the generated values as those also have a higher accumulation of values. However, with the difference that those values are located at the center of the scale or above. This observation suggests that the difference in error rate can be attributed to varying importance of the feature influences which results to less identified feature influences for the results with lower importance.

Considering the result while regarding the varied scale, which can be seen in Figure 5.9, also supports this. There are two observations: first, the error rate slightly decreases with higher scales and second, the *PeMoCo* score is also decreasing with higher scales. The error rate is therefore highest for the maximum scale 10. As we have observed in the feature importance experiments, there is a point in feature importance where *SPL Conqueror* is unable to identify the influences although these influence still have a significant impact on the error rate. In the experiments

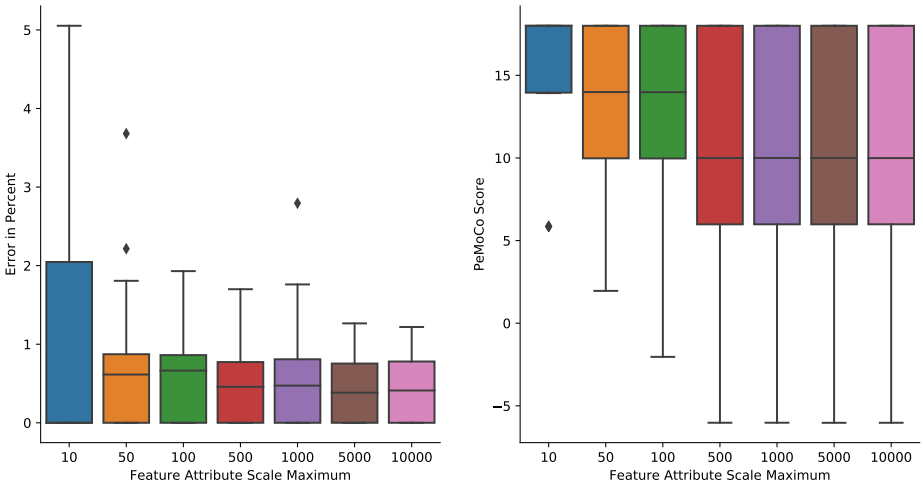

Figure 5.9: Error rate and *PeMoCo* score of the feature attribute distribution experiment considering only the scales independent of the attribute distribution

considering the influence of the feature importance, this point was roughly around an feature importance that is 15 % of the root feature. It is reasonable to assume that the higher error rate for the maximum scale 10 is due to this previously made observation as the fixed scale minimum is 1 and the accumulation of values near this minimum suggests that those are therefore roughly around that 15 % importance relation as the root feature offset is the scale maximum. Regarding higher scales, the error then decreases as those values are still close to the scale minimum of 1 and thus have a lower importance.
Some of the outliers seen in Figure 5.7 can also be explained with this observation, as those high error rates of approximately 5 % were only observed for the 15 % importance relation. This impacts the attribute distributions where the values are mainly generated at the minimum of the scale. If the values are generated at the center of the scale, the relative importance of those influences always remains the same, and therefore the error rate is minimal.
The decreasing *PeMoCo* score for higher scales can also be explained by the value accumulation near the scale. The importance of values near the minimum scale decreases if the scale maximum is increased, as the root feature value is set to the scale maximum. A lower importance leads to more unidentified influences. However, their influence on the error rate is also no longer as severe. This is also present in Figure 5.2 for the feature importance experiment where this drop in unidentified features occurs for importance values smaller 4 %.

It seems as if the main issue for feature attributes is their importance compared to the root feature. However, we could not identify further influences as the rest of the results are relatively similar.

Regarding the two sampling strategies, the results remain similar. Those results are seen in Figure 5.10 for the *feature-wise* and in Figure 5.11 for the *pair-wise* sampling. The two results with the highest error rate, namely *SWN BinarySize* and

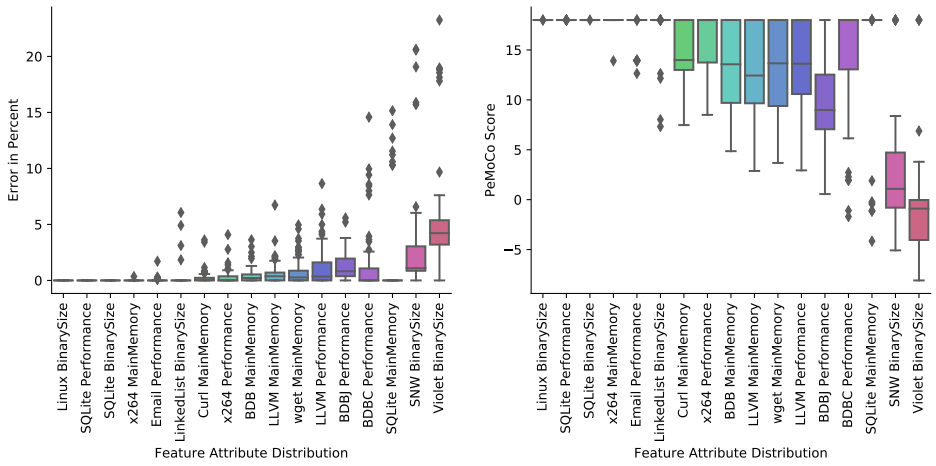

Figure 5.10: Error rate and *PeMoCo* score of the feature attribute distribution experiment considering only the attribute distributions independent of the scale with the *feature-wise* sampling strategy

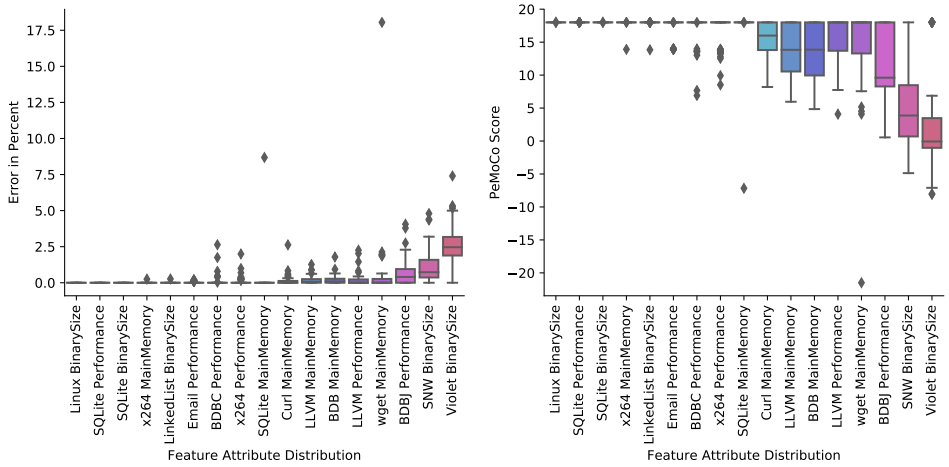

Figure 5.11: Error rate and *PeMoCo* score of the feature attribute distribution experiment considering only the attribute distributions independent of the scale with the *pair-wise* sampling strategy

Violet BinarySize are the same as in Figure 5.7. However, the overall error rate is lower for both samplings. These results further support that the influence of an attribute distribution for features is minimal and the error rate is resulting due to feature attribute values being not important enough to be identified. Therefore, for both sampling strategies, we can confirm our observations for the experiments with varying feature attribute distributions.

Experiment 2: Interaction Attribute Distribution

Considering the influence of the distribution of interaction attributes (cf. Figure 5.12), a larger variance in the error rates compared to the feature attribute distribution plot can be observed. Additionally, differently for the experiments with regards to the attribute distribution of features, a drop in the error rate does not correspond to the *PeMoCo* score, so the results with the lowest error rate also have a low *PeMoCo* score. This is also the case for the results with a high error.

Again, we need to consider the generated attribute values, seen in Figure 5.13, to be able to explain the results. The low error rate for the first three results, namely *ZipMe Performance, AJStats Performance* and *wget MainMemory*, can be explained due to their low importance as all values are near the scale minimum. However, the *PeMoCo* score could be influenced by the small variance of the generated values and not necessarily due to their importance.

Considering only results with a *PeMoCo* score lower than 15, it can be seen that those results have a small variance for the generated values and that the error rate corresponds to the mean of those values. As an unidentified interaction with a higher value has a higher influence on the error rate, this is unsurprising. This suggests that it is complicated to identify interactions that are distributed with a small variance. One explanation for this observation could be that *SPL Conqueror* can more easily find an interaction if those are more distinguishable from one another with respect to their influence. Additionally, this issue is probably amplified by the relatively high number of interactions as they are not always identified even for higher importances as seen in Section 5.2.1.

For the results of the distributions with higher variance, the error increases for a higher median of the generated values whereas the *PeMoCo* score decreases. This further confirms that those observations are not due to varying importance of the interactions as this would lead to an increased *PeMoCo* score.

We further confirmed that the seen maximum *PeMoCo* score of 34 is actually a score from a result where all interactions have been identified.

Finally, to rule out other influences, the results while solely regarding the different scales can be seen in Figure 5.14. Apart from the random factor given due to the implementation, there can not be seen much difference in the results, thus, the observed results are not overly influenced by the scale.

Considering the *pair-wise* sampling, seen in Figure 5.15, the overall results are erratic. We assume that this is due to the high number of interactions that for the *pair-wise* sampling lead to difficulties identifying feature influences, especially if the interaction importance is high. This sampling is designed to identify interactions

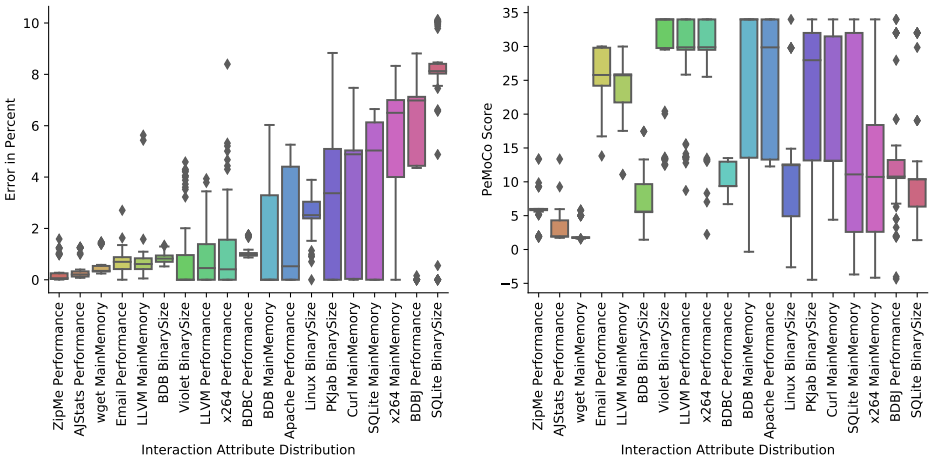

Figure 5.12: Error rate and *PeMoCo* score of the interaction attribute distribution considering only the attribute distribution independent of the scale

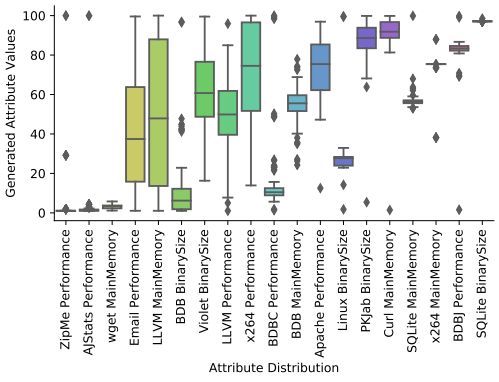

Figure 5.13: Generated interaction attribute values for a scale maximum of 100

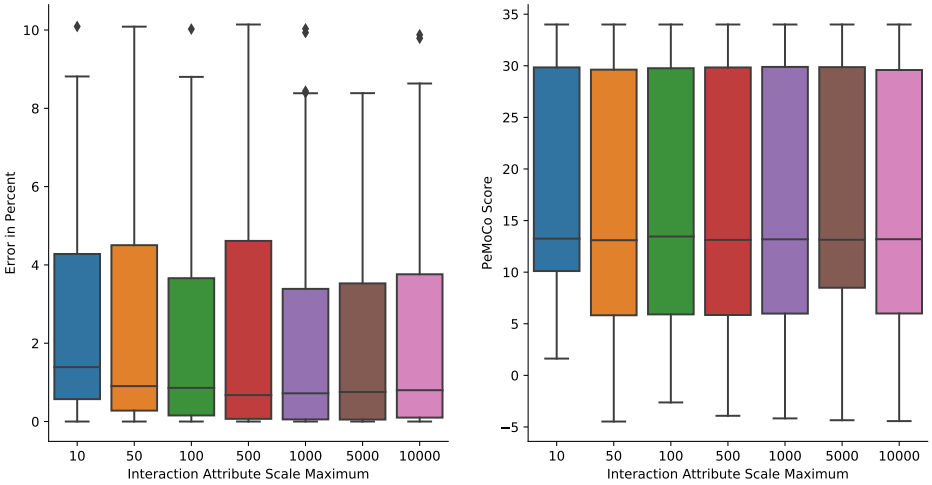

Figure 5.14: Error rate and *PeMoCo* score of the interaction attribute distribution experiment considering only the scale independent of the attribute distribution

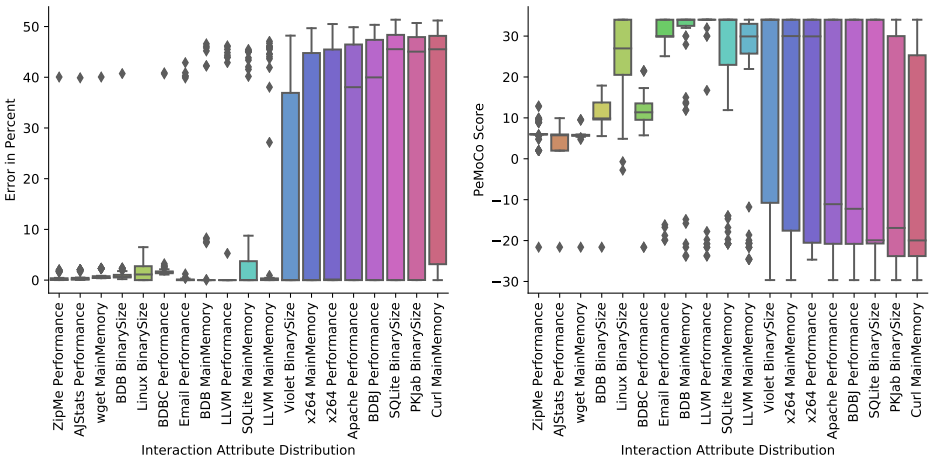

Figure 5.15: Error rate and *PeMoCo* score of the interaction attribute distribution experiment considering only the attribute distribution independent of the scale with the *pair-wise* sampling strategy

of order 1, however, it could be that difficulties to identify influences of features arise if the number of interactions is too high. This is also supported by the in Section 5.2.1 evaluated results of the experiments with regards to numbers of interactions with *pair-wise* sampling. For the results with lower error rates, it can be again observed that the three results with low interaction values, namely *ZipMe Performance, AJStats Performance* and *wget MainMemory*, have negligible influence and are not identified as the *PeMoCo* score indicates. However, for almost every attribute distribution, an outlier for an error rate of 40 % exists, which in turn supports that this is a general issue for the *pair-wise* sampling.
As *feature-wise* sampling is unable to identify interactions, the resulting plot can be found in the Appendix.

5.2 Effects of Interaction Number and Order Distribution

Again, the results for the following experiments are presented according to the experiment separation regarding to **RQ 2.1** and **RQ 2.2**. In general, for the following experiments, we refrain from using the *PeMoCo* score due to the in Section 5.3.1 discussed challenges and use the percentage value of found interactions instead. Thus, the following figures for both experiments again show two plots, with the error rate in the left plot and the percentage value of found interactions in the right plot.

5.2.1 Interaction Numbers and Orders

In relation to **RQ 2.1**, this experiment attempts to find influences on the learnability with regards to varying interaction number and interaction order. The plots for this experiments either show the results for the varied number of interactions or the varied order of interactions. If not further specified, considering the number of interactions, the shown data points are the results from the 10 repetitions for each order of interactions and vice versa. However, it should be noted that we chose to ignore order 6 completely for the shown plots as it only has a maximum interaction number of 8.

Regarding the number of interactions, we can see that for an increasing number of interactions, the error rate gets higher as well as a lower percentage interactions is identified (cf. Figure 5.16). The increase in interactions leads to a linear increase in error rate whereby it has to be taken into account that the visual exponential error increase is only due to the selection of tested interaction numbers. The increased error rate is corresponding to the decreased percentage of found interaction. Additionally, it can be observed that for an interaction count greater than 30, *SPL Conqueror* is not able to identify every interaction. This might be due to the higher required order as those interaction counts are only possible in higher order.

Considering the order (cf. Figure 5.17), the amount of found interactions decreases for higher orders as expected – but the observed error rate is not completely corresponding to this. The mean error rate is nearly the same for order 2 and 3 as well as order 4 and 5. If we examine the results of varying order while limiting the number of interactions to solely 25 (cf. Figure 5.18), as this is the maximum tested number supported by the majority of orders, we can again see this decrease

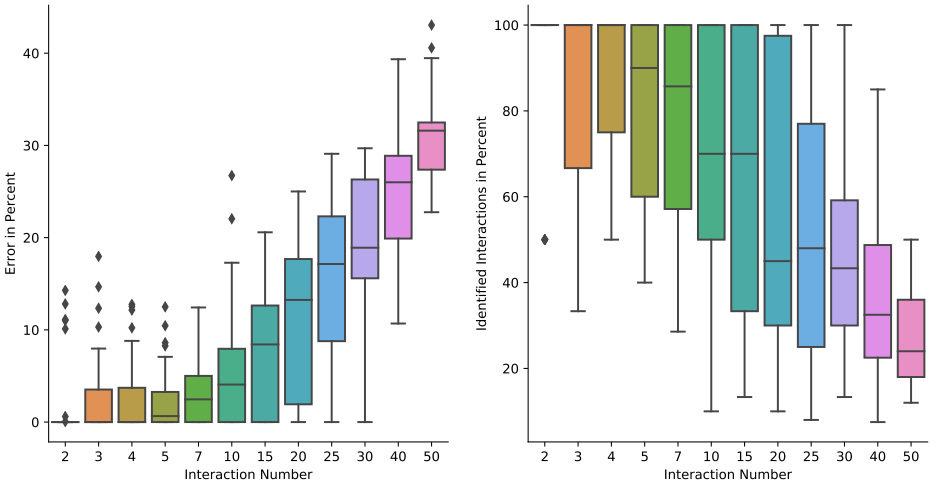

Figure 5.16: Error rate and identified interactions of the interaction number experiment considering only the interaction numbers independent of the order

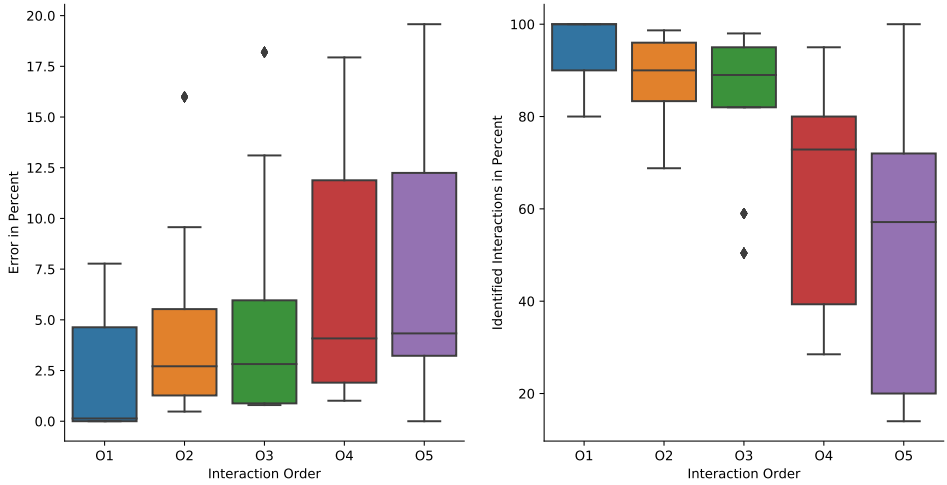

Figure 5.17: Error rate and identified interactions of the interaction number experiment considering only the interaction order over all interaction numbers up to 25

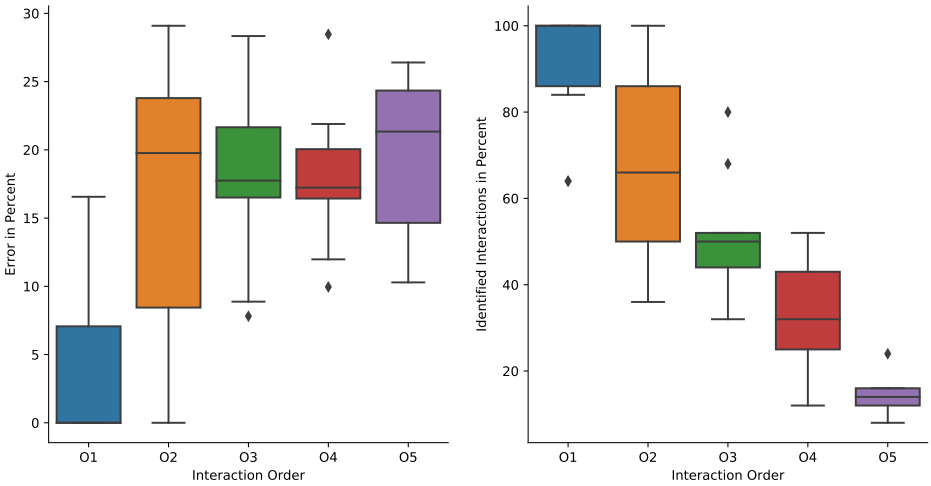

Figure 5.18: Error rate and identified interactions of the interaction number experiment considering the interaction order and only an interaction number of 25

in identified interactions. However, the error rate does not increase correspondingly: it even slightly decreases from order 2 to order 4. This can be explained as the influence of interactions on all configurations decreases with increasing order. This connotes that an interaction of higher order is present in less configurations than an interaction of a smaller order which in turn leads to the lower error rate. This effect appears in the decreasing error rate for higher orders. This effect also explains why for interaction numbers greater than 30, *SPL Conqueror* never identifies all interactions.

Considering the two sampling strategies, all plots are found in the Appendix as they both can either find only interactions of order 1 or no interactions at all and therefore the results are mostly monotone. However, for the *pair-wise* sampling in Figure 5.19 an additional figure is given, showing the results for varying number of interactions while only considering an interaction order of 1. It can be seen that the overall error is high as well as that only for an interaction number of two all interactions are always found. However, this is specific to the *pair-wise* sampling and is only included to better explain the results for the experiments with varying attribute distributions of interactions while using *pair-wise* sampling in Section 5.1.2 as those showed erratic results (cf. Figure 5.15).

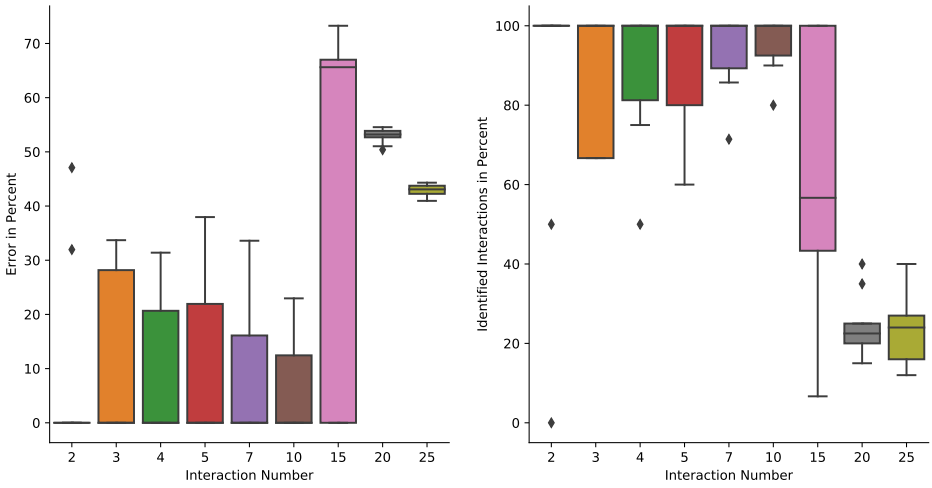

Figure 5.19: Error rate and identified interactions of the interaction number experiment considering the interaction number and only interactions of order 1 with the *pair-wise* sampling strategy

5.2.2 Interaction Order Distribution

With this experiment, we want to identify possible influences that arise due to the varying distribution of a number of interactions over multiple orders. Therefore, the following plot shows the tested interaction distribution over orders on the x-axis. Seemingly counter-intuitive, the error rate of the *ascending* interaction distribution is better than the *descending* distribution (cf. Figure 5.20). We expected that the generated interactions of the *descending* interaction distribution are easier to identify as they are realistically distributed and therefore lead to a better error rate. This is only partly the case, as the number of found interactions is actually better than the *ascending* distribution, but the error rate is worse.

For the *ascending* interaction distribution the *performance-influence model* is worse, the average error rate, however, is better. In this case, those interactions are not identified, but must have minor influence on the error rate. However, regarding the *descending* interaction distribution, more identified interactions are leading to a higher *PeMoCo* score. As the error rate is higher, the unidentified interactions must have a higher impact. This effect was previously observed in Section 5.2.1 where interactions of higher orders had a smaller effect on the error rate.

As for the other three interaction distributions, the normal distribution has the worst error rate, as well as the least amount of identified interactions. This order distribution has most interactions in order 3 which means that those will be found less likely, however, those interactions still have an effect on the error rate. This can also be seen in Figure 5.18 where interactions in order 2 and 3 have the highest error rate. The *invers-normal* distribution has the best overall error rate as well as the second best average amount of identified interactions. Normally 11 out of 25 interactions are correctly identified as the median percentage of identified interactions is 45 %. Therefore, most of the interaction of order 1 and order 2 must be detected as lower order interactions are more easily found. In comparison to the other distributions,

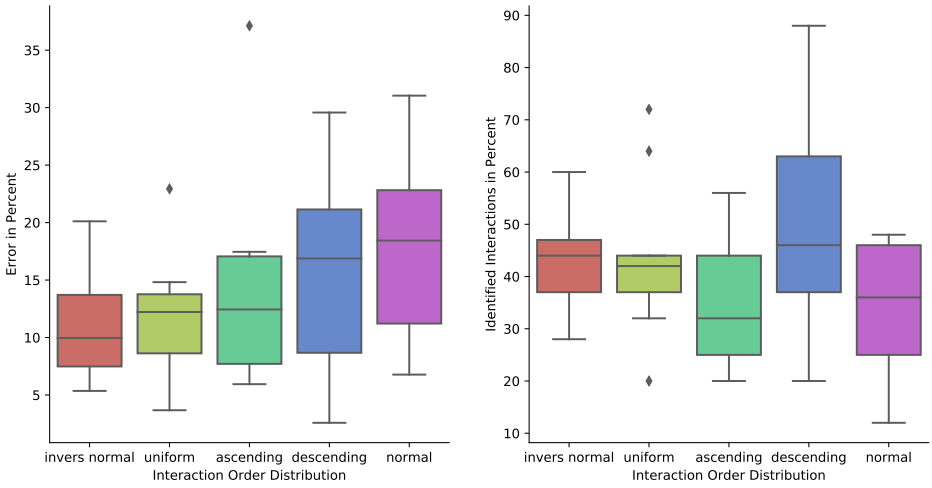

Figure 5.20: Error rate and identified interactions of the interaction order distribution experiment

this distribution has not many interactions in order 3 and 2 where their impact on the overall error is still high, thus, leading to a lower error rate.
The error of the uniform distribution is slightly higher and there are also less interactions identified. As those interactions are again probably found in lower orders, this error can again be attributed to the unidentified interactions of order 3.

As both sampling strategies are unable to identify interactions of higher order, the resulting plots for those samplings are found in the Appendix.

5.3 Discussion

In this section, the presented results will be summarized and the research questions from Chapter 1 will be answered. Additionally, we outline some challenges with *PeMoCo* that arose while evaluating the experiments where higher order interactions were incorporated.

5.3.1 Challenges with PeMoCo

As mentioned in Section 2.2, *SPL Conqueror* does not perform a backwards elimination of influences. Thus, already identified influences will not be eliminated if those influences become irrelevant in future rounds. Additionally, *SPL Conqueror* only combines existing influences and the present features of the feature model. Thus, to find an interaction of order 2 or higher, a corresponding sub-interaction has to be found primarily and those sub-interactions will always remain an influence in the model. Regarding experiments with higher interaction numbers, this implies that the output model will always contain more terms than present in the input. As a concrete example, the best possible output from *SPL Conqueror* for a generated model with an interaction of order 3 would always contain 2 sub-interactions, of order 1 and 2 respectively, both with an influence of nearly 0 and the order 3 interaction that is actually present.

As the *PeMoCo* score is mainly determined by the terms that are present in both models, the score even for a perfectly learned model under those constraints is not optimal. Furthermore, it is also possible that the score of a result, where only the feature influences were identified, might be better than the score of a result where all interactions were identified, due to a large amount of sub-interaction terms.
Therefore, the *PeMoCo* score is unsuitable for higher orders as it can not identify terms with no influence. As a consequence, this score can not be utilized. However, we included the amount of correctly identified interactions and used those to interpret the observed results even though this is incapable to completely depict the correctness of the identified model.
This change to our considered dependent variable is conform with the definition of learnability as this was the capability to correctly identify an accurate model. The accuracy is still given through the error rate and the correct model is given through the found interactions as for those experiments the feature influences are mostly not important.

5.3.2 Attribute Values and Distribution

RQ 1.1: How does a change in the relative importance of configuration options affect the learnability?

Based on the results explained in Section 5.1.1, the observation was made that only influences are identified if they are important enough. Therefore, if the importance is too low, learnability is negatively influenced as the produced model is no longer completely accurate. There is also an influence on the error rate which is a direct result of the produced model. However, if the importance is further reduced the influence on the error rate can be ignored. As those influences have no influence apart from being not identified, those can probably be neglected.
Concluding, there is a critical range for the importance of influences where the learnability is effected negatively. Outside that range, either if there is no more influence on the error rate or if all influences are easily identified, there is no observed effect.
To avoid overfitting the dataset, a lower number of terms in a model is preferred. Thus, for both importance experiments, this behaviour is desirable. If a term would be present for every point of data, the learning results are prefect, however, the model is not generally applicable [Rob14]. Therefore, this error is justified and will always occur as unimportant terms need to be avoided.

RQ 1.2: How does a change in the distribution of influences of configuration options affect the learnability?

As the results explained in Section 5.1.2 show, the effect of different attribute distributions varies. For feature influences, the effect is mostly small and can mainly be attributed to a small feature importance. This is also supported by the experiments with regards to the importance of features. Additionally, the overall low error rate leads to the conclusion that the feature attribute distribution has no effect, if the generated values are important enough that they are found.
However, for interactions only the error can be attributed to the importance. The ability to identify those interactions depends on the attribute distribution. If the

interaction attribute values are similar, the learnability is effected negatively as they are less likely to be identified. For distributions where the accumulation of values is less, the error could be attributed to the importance.
For both distribution experiments, we were unable to completely explain all observations and further experiments are needed.

5.3.3 Interaction Numbers and Orders

RQ 2.1:How does the number and order of interactions influence the learnability?

As the results in Section 5.2.1 show, regarding higher interaction numbers, there is a clear increase in error rate as well as a limited capability to identify those interactions. Even in this scenario, where all settings for this experiments were chosen to enable finding those interactions, this is unambiguous. Therefore, the learnability is always negatively influenced by an increasing number of interactions.
However, this is not entirely correct for the order due to the varying importance of those influences. As an interaction of higher order is present in less configurations, its influence on the error rate is reduced. Therefore, even if the number of unidentified interactions increases linear for higher orders, the error rate does not. We even observed lower error rates compared to the next lower order, but those can also be attributed to the random factor of the data generation.
Additionally, this reduced influence of higher orders does not imply that those are always unidentified as the results for small interaction numbers show.

RQ 2.2: How does the distribution of interactions over varying order influence the learnability?

Based on the results in Section 5.2.2, the distribution of interactions over different orders has an influence on the learnability mainly via the amount of interactions for specific orders. The error rate decreases if more interactions exist in order 2 and order 3 and the number of found interactions decreases if the order is higher. Those influences on the learnability are according to the results of the interaction number experiments in Section 5.2.1. However, it is noticeable that those orders are also the orders that support the highest numbers of interactions. A clear influence that could be directly attributed to the varying distribution was not observed. Therefore, this research question can not be entirely answered. There are further implications with regards to realistic interactions for those experiments which are discussed in Chapter 7.

6. Threats to Validity

In this chapter the possible threats to both internal and external validity are discussed.

6.1 Internal Validity

Internal validity signifies to what extend our conclusions are valid and how systematic errors were avoided.
For our axillary implementation, meaning the automation of the data generation and starting the experiments, the correctness was confirmed as only the generated experiments needed to be checked. The main potential error was the computation of the generated measurements that are needed as *SPL Conqueror* input. As the output of *Thor* consists of the influences of both features and interactions, the NFP value for each configuration needs to be calculated. We spot-checked the results manually and could confirm that the computation of a NFP value is correct even with included interactions. Additionally, we verified that for all experiments with interactions, those occur in the generated data. Moreover, the additional implementations to *Thor* needs to be validated. Generally, despite the existence of some software bugs, we expect *Thor* to be working correctly. Therefore, confirming correctness for both the additions that enabled existing functionality for the command-line as well as the bug corrections was simply done via checking if for a sample the generated result was as expected. This is also justified as the bug corrections are simple changes that are only effecting the concerned method.
Correctness was also confirmed for the new functionality added to *Thor* via sample experiments. Those functions in *Thor* solely ensure that its results are more suitable, thus increasing the validity of *Thor*. Regarding the random function addition the most suitable random sample out of 100 is now chosen, opposed to a single random sample. And for the scaling addition we modify a random sample in a way that only eliminates values that are not necessary for a correct result. The correctness check is still conducted afterwards, also guaranteeing a valid result.
Concerning the validity of our experiments, we first avoided the bias that can occur due to the random generation of values as we repeated every experiment 10 times.

Additionally, we can guarantee for all experiments that there is a causal connection between the observed results and our independent variables. For every experiment, as many influences as possible were eliminated and as we only have a finite set of possible influences on the generated data as well as the *SPL Conqueror* settings, there are no changing variables that could influence our results.

6.2 External Validity

External validity denotes to what extent our conclusions can be generalized.
In general, *Thor* is mostly able to create realistic data and therefore, results utilizing this realistic data generation can more easily be generalized. Furthermore, all conducted experiments are of a very basic kind. All possible influences on the results were reduced to a minimum. The used feature model contained no constraints, and so every feature was optional. The observed results can therefore be only attributed to the varied inputs. As the observed influences on the learnability can only be due to the input and also valid in the most basic scenario, the are generally valid.
The used feature model only contained 8 features, however, this number is sufficient as we wanted base-line results. Additionally, this is only an issue for the experiments with varying attribute distribution as only those could benefit from a higher number of features. However, the observed generated values do vary largely, meaning that even for 8 features the utilized attribute distributions largely effect the data.
As mentioned in Section 2.3, *Thor* can not generate realistic interactions as they are chosen randomly and new interactions are therefore not based on pre-existing smaller interactions. This is solely a problem for the interaction order distribution experiments as we, apart from those, only have interactions of order one or explicitly want to see the effect of interactions of higher orders with no interactions in lower orders. However, for a software system where interactions are more according to real world observations, *SPL Conqueror* is more likely to identify those interactions as explained in Section 5.3.1. However, for those realistic interaction order distributions, this was already observed.
Finally, the basic kind of the conducted experiments lead to a high confidence in the validity of our results.

7. Future Work

This chapter describes possible extensions to the experiments as well as further scopes to continue the work done in this thesis.
For this thesis, we resorted to only using a single feature model - this selected feature model could also be a factor on the learnability of *SPL Conqueror* and therefore also of interest. However, the focus of this thesis was the attribute values of influences and not the influence of varying feature models, but this aspect is also worthy of investigation.
Another future work would be considering the influence of negative attribute values as we mostly avoided those. As *SPL Conqueror* can not manage negative configurations, there is the need to find a way that allows negative values, but also efficiently prevents negative configurations. This challenge needs to be solved to be able to effectively evaluate negative attribute values. Negative attribute values were incorporated in the feature importance experiments, but they did not have an influence on those specific observed results. However, this may not be generally applicable and therefore needs to be further investigated.
Additionally, there is also the aspect of time to accuracy that was not evaluated. Meaning the time that *SPL Conqueror* needs to learn a *performance-influence model* that qualifies for a certain accuracy. For the conducted testing, this was not a concern as the focus was general on the ability to learn a model and not the needed time. To be able to evaluate this aspect, the feature model also needs to be changed to a larger model as *SPL Conqueror* is always fast on the selected small model and will have only a small variance in the time results.
Furthermore, there is the need to better categorize and control the selected attribute distributions. As the present distributions were real world samples, we had little information about their characteristics and the results of the feature attribute experiments were not too extensive and we were only able to draw basic conclusions about the influences of attribute distributions. Expanding those experiments with better controlled attribute distributions with exactly known characteristics might help to be able to draw more comprehensive conclusions.
As the interactions generated by *Thor* were always random combinations of features, we were unable to evaluate the effects of realistic interactions. To this end,

we need to influence the interaction generation over multiple orders to create related interactions where for every interaction of higher order a sub-interaction is present. Additionally, there is the need to influence how many interactions are in such a sub-interaction relationship to each other interactions. This is also relevant for the interaction order distribution experiments. Those distributions may have a bigger impact and a more identifiable influence on the learnability if the interactions are in a more realistic way.
Finally, as the *SPL Conqueror* settings were not changed, those could also have a influence that needs to be evaluated. Furthermore, another aspect is that for different software system, different settings may be superior. We have also observed this for the varying sampling strategies we utilized. However, evaluating the *SPL Conqueror* settings was out of the scope of this thesis and is therefore another future work.

8. Conclusion

In this thesis, we were able to evaluate the learnability of software systems in a way that is unachievable without the possibility to use generated test data.
We aimed at finding influences on the learnability of a software system. To this end, we designed experiments tailored to examine the effects of certain characteristics that we could influence through *Thor*. The varied characteristics are the importance of influences, different attribute distribution of the influence values, number and order of interactions and also the distribution of interactions over varying orders.
For those conducted experiments, we found that there is a critical point if the importance of an influence is reduced, it is no longer identified by the machine learning approach while it still influences the error rate. This behaviour, however, is justified as it avoids overfitting the training data. For the interaction number and order experiments, we found that only increasing the number inevitably has a negative influence on the learnability of a software system. By contrast, the order does not necessarily lead to an increase in error rate and only the capability to identify all influences is negatively impacted. Regarding the experiments for varying attribute distributions, we observed that no explicit influence could be attributed to the attribute distribution of features. However, for the attribute distributions of interactions, we observed that interactions with a lower variance in attribute values had a negative impact on the learnability. Nevertheless, we were unable to completely explain all observations. Finally, regarding the experiments for varying the distribution of interactions over different orders, we again can not completely identify clear influences on the learnability. However, we observed that distributions where less interactions exist in orders that support a higher total number of interactions, performed better with regards to the learnability.
As the state of *Thor*, especially the command-line, was suboptimal, conducting those experiments was more time consuming than expected. This lead to a reduction in the total amount of possible experiments as well as their extent. Nonetheless, based on this thesis, some significant software bugs were identified in *Thor* as well as functions added that will ease future use of this tool.
Even though we were not completely able to answer all research question, our identi-

fied influences on the learnability will help in understanding the capabilities of *SPL Conqueror* as well as be supportive to explain possible obstacles to the learnability of configurable software systems. However, there is still additional work to be done as some aspects were not completely discussed.

Bibliography

[ABKS16] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. *Feature-oriented software product lines*. Springer, 2016. (cited on Page 5)

[AKS+13] Sven Apel, Sergiy Kolesnikov, Norbert Siegmund, Christian Kästner, and Brady Garvin. Exploring feature interactions in the wild: the new feature-interaction challenge. In *Proceedings of the 5th International Workshop on Feature-Oriented Software Development*, pages 1–8. ACM, 2013. (cited on Page 7)

[BTRC05] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortés. Automated reasoning on feature models. In *International Conference on Advanced Information Systems Engineering*, pages 491–503. Springer, 2005. (cited on Page 6)

[CS14] Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014. (cited on Page 7)

[GSKA16] Alexander Grebhahn, Norbert Siegmund, Harald Köstler, and Sven Apel. Performance prediction of multigrid-solver configurations. In *Software for Exascale Computing-SPPEXA 2013-2015*, pages 69–88. Springer, 2016. (cited on Page 1)

[Kal16] Christian Kaltenecker. Comparison of Analytical and Empirical Performance Models: A Case Study on Multigrid Systems. Masterthesis, University of Passau, Germany, 2016. (cited on Page 10)

[KSK+18] Sergiy Kolesnikov, Norbert Siegmund, Christian Kästner, Alexander Grebhahn, and Sven Apel. Tradeoffs in modeling performance of highly configurable software systems. *Software & Systems Modeling*, pages 1–19, 2018. (cited on Page 8)

[Leu16] Thomas Leutheusser. Generating Realistic Attributed Variability Models. Masterthesis, University of Passau, Germany, 2016. (cited on Page 8)

[Rob14] Christian Robert. *Machine learning, a probabilistic perspective*. Taylor & Francis, 2014. (cited on Page 45)

[SGAK15] Norbert Siegmund, Alexander Grebhahn, Sven Apel, and Christian Kästner. Performance-influence models for highly configurable systems. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 284–294. ACM, 2015. (cited on Page 1 and 7)

[Sie12] Norbert Siegmund. *Measuring and Predicting Non-Functional Properties of Customizable Programs*. PhD thesis, University of Magdeburg, Germany, 2012. (cited on Page 6)

[SKK+12] Norbert Siegmund, Sergiy S Kolesnikov, Christian Kästner, Sven Apel, Don Batory, Marko Rosenmüller, and Gunter Saake. Predicting performance via automated feature-interaction detection. In *Software Engineering (ICSE), 2012 34th International Conference on*, pages 167–177. IEEE, 2012. (cited on Page 7 and 8)

[SSA17] Norbert Siegmund, Stefan Sobernig, and Sven Apel. Attributed variability models: outside the comfort zone. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 268–278. ACM, 2017. (cited on Page ix, 2, and 9)

[XJF+15] Tianyin Xu, Long Jin, Xuepeng Fan, Yuanyuan Zhou, Shankar Pasupathy, and Rukma Talwadker. Hey, you have given me too many knobs!: understanding and dealing with over-designed configuration in system software. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 307–319. ACM, 2015. (cited on Page 1)

A. Appendix

All additionally produced plots are shown in this chapter. Furthermore, all conducted experiments as well as the used Python scripts and tools are provided digitally by the included DVD.

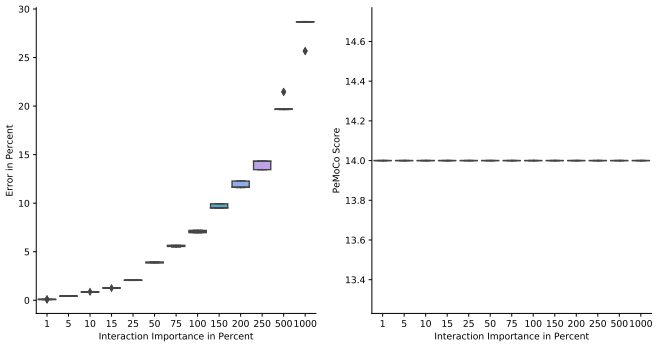

Figure A.1: Error rate and *PeMoCo* score of the interaction importance experiment with the *feature-wise* sampling strategy

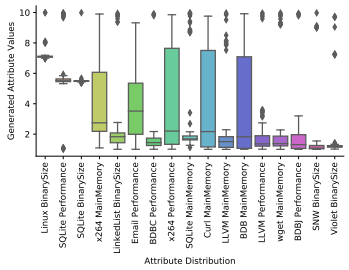

Figure A.2: Generated feature attribute values for a scale maximum of 10

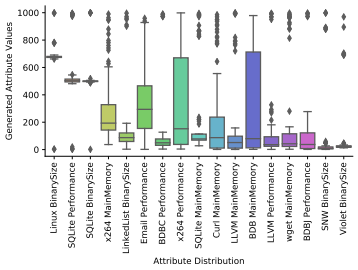

Figure A.3: Generated feature attribute values for a scale maximum of 1000

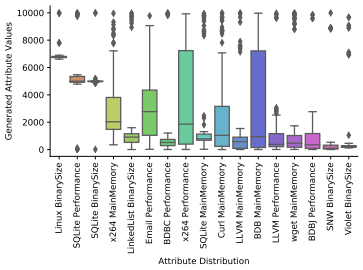

Figure A.4: Generated feature attribute values for a scale maximum of 10000

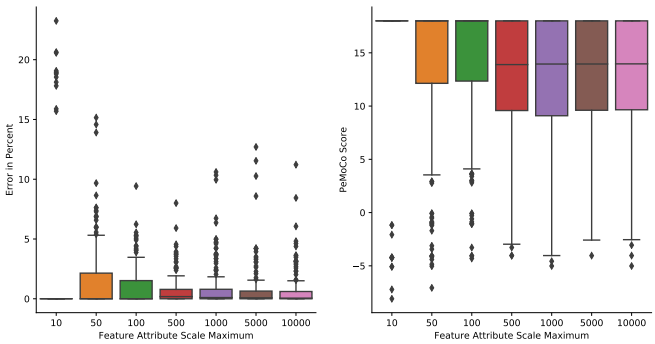

Figure A.5: Error rate and *PeMoCo* score of the feature attribute distribution experiment considering only the scales independent of the attribute distribution with the *feature-wise* sampling strategy

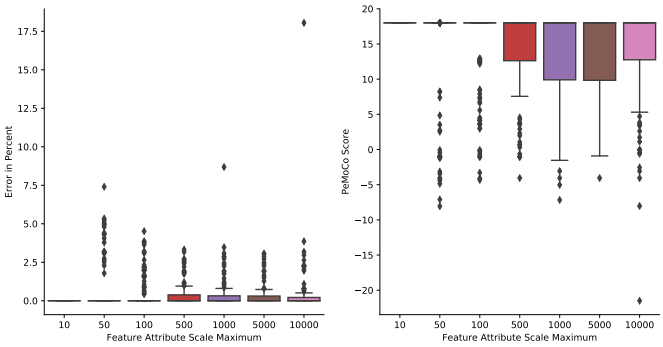

Figure A.6: Error rate and *PeMoCo* score of the feature attribute distribution experiment considering only the scales independent of the attribute distribution with the *pair-wise* sampling strategy

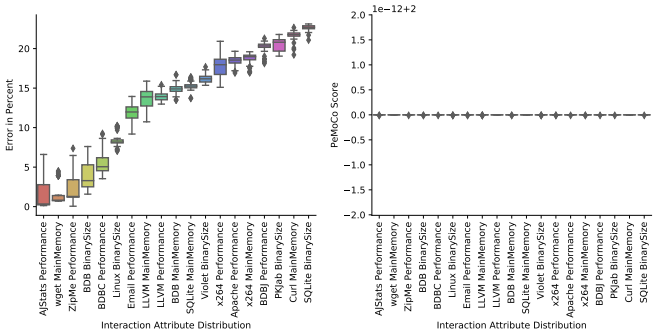

Figure A.7: Error rate and *PeMoCo* score of the interaction attribute distribution experiment considering only the attribute distribution independent of the scale with the *feature-wise* sampling strategy

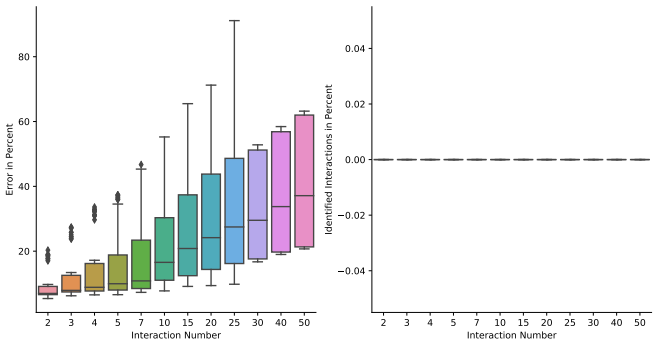

Figure A.8: Error rate and found interactions of the interaction number experiment considering only the interaction numbers independent of the order with the *feature-wise* sampling strategy

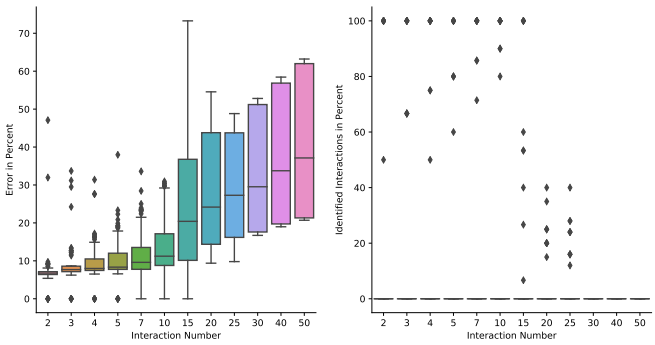

Figure A.9: Error rate and found interactions of the interaction number experiment considering only the interaction numbers independent of the order with the *pair-wise* sampling strategy

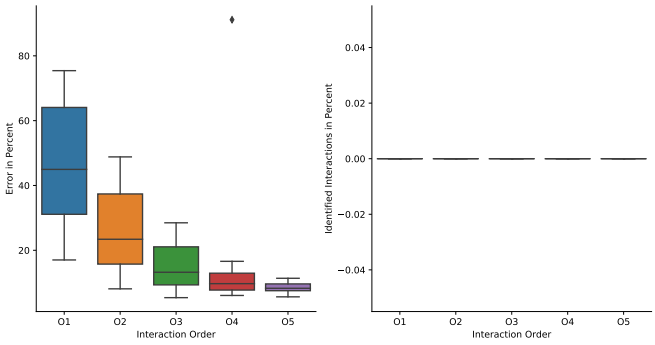

Figure A.10: Error rate and found interactions of the interaction number experiment considering only the interaction order and all interaction numbers up to 25 with the *feature-wise* sampling strategy

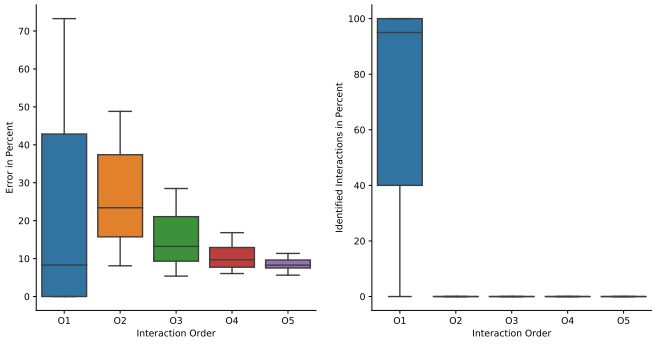

Figure A.11: Error rate and found interactions of the interaction number experiment considering only the interaction order and all interaction numbers up to 25 with the *pair-wise* sampling strategy

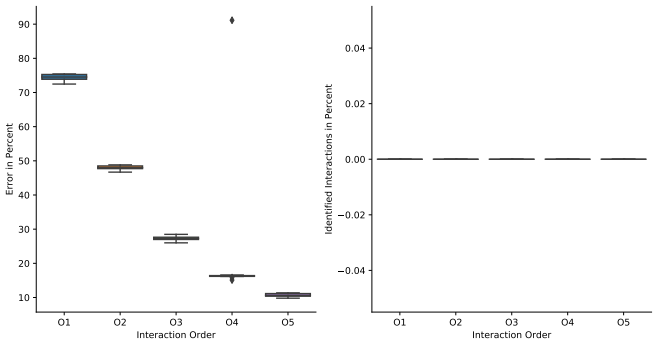

Figure A.12: Error rate and found interactions of the interaction number experiment considering the interaction order and solely an interaction number of 25 with the *feature-wise* sampling strategy

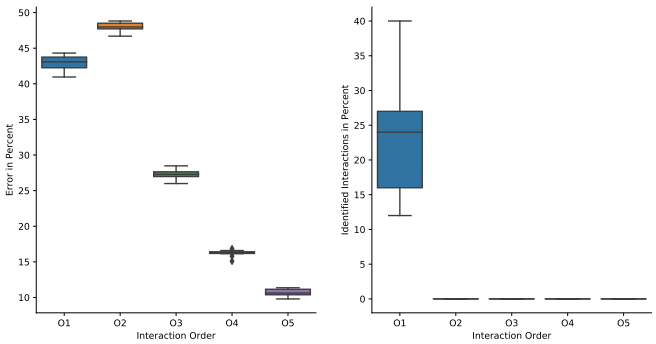

Figure A.13: Error rate and found interactions of the interaction number experiment considering only the interaction order and solely an interaction number of 25 with the *pair-wise* sampling strategy

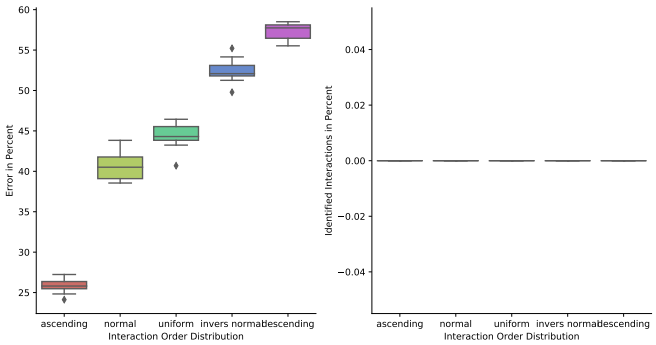

Figure A.14: Error rate and identified interactions of the interaction order distribution experiment with the *feature-wise* sampling strategy

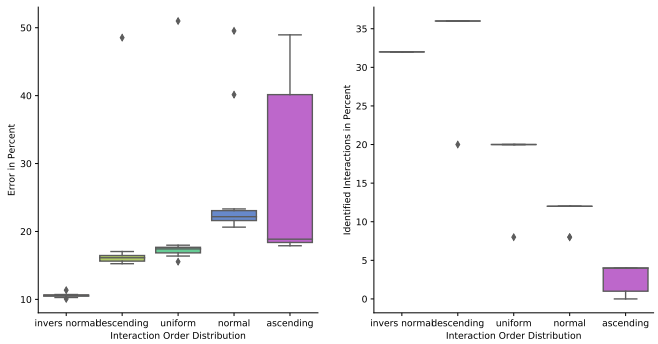

Figure A.15: Error rate and identified interactions of the interaction order distribution experiment with the *pair-wise* sampling strategy

Eidesstattliche Erklärung:

Hiermit versichere ich an Eides statt, dass ich diese Bachelorarbeit selbständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe und dass alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind, sowie dass ich die Bachelorarbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

Jakob Hoffmann

Passau, den 25. Juni 2018