University of Passau

Department of Informatics and Mathematics

Master's Thesis

Generating Realistic Attributed Variability Models

Author:

Thomas Leutheusser

September 28, 2016

Advisors:

Prof. Dr. Sven Apel
Chair of Software Engineering

Dr. Norbert Siegmund
Chair of Software Engineering

Leutheusser, Thomas:
Generating Realistic Attributed Variability Models
Master's Thesis, University of Passau, 2016.

Contents

List of Figures

List of Tables

List of Acronyms

CmV	Cramer von Mises
EA	Evolutionary Algorithm
KS	Kolmogorov-Smirnov

1. Introduction

Recent movements in software development focus on support for multiple platforms and devices to be applicable for the largest possible group of systems. The development of these cross-platform applications is a time intensive task since single components have to be implemented for different platforms. Additionally lots of software products grow over time and add more functionality, most of which are not even utilized by the whole user base. To fulfill requirements such as performance or memory consumption it is desirable to deliver custom tailored products, including only the desired functionality. This advances the need for configurable software systems with reusable components to reduce the implementation effort and cost while increasing the maintainability.
Different programming paradigms have been established to achieve the previously mentioned goals. Feature-oriented software development (FOSD) is such a paradigm which focuses on the composition of features for the creation of variable software systems. The variability of a software system in terms of configuration options is captured in a variability model. Feature modeling has been proven successful in both research [CGR$^+$12] and industry [BRN$^+$13] ,since their first introduction in the feature-oriented domain analysis (FODA) method by Kang [KCH$^+$90]. A variability model specifies all valid variants of a configurable system in a compact representation with selectable configuration options and constraints among them. The variation in a selection of configuration options implies differences in the quality concerns of a system such as performance, main memory usage, energy consumption, security or footprint. To name just one example: An encrypted database increases the security in contrast to a normal database, but might have negative impact on the performance.
An extension to the variability model, the attributed variability model, where quality attributes are annotated directly to the features, allows the reasoning about non-functional properties of a generated product.
Various studies containing attributed variability have been published and their main background are multi-objective optimization, runtime-adaptation or non-functional property prediction. Variability models with realistic attribute values are however very rare in the literature, because the measurement of non-functional properties

from variants and the impact of individual features on that quality attribute is expensive. It requires the building of products, taking measurements and finally identifying the impact each feature has on the measured property. These tasks are infeasible for medium and large sized software systems since their effort scales exponentially with the amount of features.
Therefore the studies tend to create their own variability models with randomly generated attribute distributions like the normal or uniform distribution. Furthermore feature interactions are largely ignored in the literature, despite their evidenced existence and influence on non-functional properties in almost every real world system. This poses several problems. On the one hand, synthetically generated value distributions differ largely from the ones found in real-world systems. These attribute density distributions contain large gaps and strong peaks, opposed to the smooth curve of a normal or uniform density distribution. This influences the computational complexity of algorithms working on value distributions. For example, the search for an optimal configuration with a genetic algorithm might favor a simple distribution with only one local maximum over the uneven distributions of real-world systems with multiple local maxima.
On the other hand, ignored feature interactions can lead to unexpected changes in the non-functional properties of a product. This reduces the accuracy and even the correctness of algorithms, as the presence of a feature interaction might alter a calculated optimal configuration to a suboptimal one in a deployed system and vice versa.
The lack of realistic attributed variability models and the negligence of feature interactions is problematic for the research community. It could guide the researchers into false assumptions due to the usage of synthetic attribute distributions with different characteristics to real-world systems.

1.1 Goal of this Thesis

To overcome the shortage of realistic attributed variability models, we propose a tool for the creation of attributed variability models, with realistic configuration option attribute values, realistic system variant value distributions and including real-world feature interaction values.
To achieve this task several sub-steps are necessary.
First we analyze the usage of attributed variability models in the current literature. We examine the following topics and questions.

- Sources of attributed variability models: Are the models used in current literature derived from real-world systems or synthetically generated models?
- Initial generation of synthetic attributes: Are researchers using artificially generated or real-world attribute distributions in their variability models?
- Feature interactions: Are researchers considering feature interactions in their publications?
- Research domain: Which problems are solved in the publications?

Secondly we extract the attribute values of configuration options, interactions and variants from the SPLConqueror[SRK+12] dataset, which is currently the only publicly available dataset. We visualize the data from different measured real-world systems for the non-functional properties performance, binary size and main memory consumption. Then we identify patterns in the distributions and point out the differences to synthetically generated distributions.
Lastly we implement a tool which inputs are:

- A non-attributed variability model
- A configuration option attribute distribution from the SPLConqueror dataset or own values
- An interaction value distribution from the SPLConqueror dataset or own values
- The amount and degree of interactions to be woven in
- A variant value distribution from the SPLConqueror dataset or own values

The tool's output is an attributed variability model, where the feature-, interaction-, and variant distributions are optimized to resemble the selected input distributions as close as possible.
At the core a genetic algorithm searches for the pareto-optimal solutions to the multi-objective optimization problem, where the fitness is defined over the similarity of configuration option-, interaction-, and variant distribution. The goodness-of-fit of two distributions can be calculated with different statistical tests or distance metrics. We implemented two statistical tests and two distance metrics and provide an evaluation of these test statistics in performance and quality.
In particular we try to answer the following research questions for the literature study (LS), SPLConqueror dataset analysis (DA), and the tool implementation (TI)

LS RQ1:	From the papers that use attributed variability models, how many use synthetic attribute values and how are they generated?
LS RQ2:	Are feature interactions considered in the literature?
DA RQ1:	Which patterns of attribute distributions are present in real-world systems for configuration options, interactions and system variants?
TI RQ1:	How well do the test statistics operate in performance and quality over different attribute-value distributions and problem sizes? Is there a single best metric?
TI RQ2:	Is there an overall best variant sampling heuristic? Is there an optimal trade off between the amount of generated variants and the accuracy to the original variability model?

1.2 Structure of the Thesis

Chapter 2 gives an overview of the fundamental components used in this thesis.
The results of the literature survey are provided in Chapter 3:Literature study.

Chapter 4:Problem Statement explains the major problems we needed to overcome and Chapter 5:Approach describes the proposed solutions to the mentioned problems. Chapter 6:Evaluation explains several experiments and a subsequent discussion of the results. Chapter 7:Related Work and the last chapter Conclusion close the thesis.

2. Background

In this chapter, we provide an outline of the key concepts that are used in our generator.
The first section gives a brief explanation of attributed variability models and feature interactions. Section 2.3 introduces a statistical resampling technique called kernel density estimation. Section 2.5 shows the main components of genetic algorithms and lastly we describe different test statistics and metrics for the comparison of distributions in Section 2.6

2.1 Variability Models

Configurable software systems can be described by the set of valid variants and commonalities between them. A variability model is a representation of all possible configuration options, their relationships and constraints among them. These relationships and constraints reduce the set of possible configuration options to the subset of valid configurations.

The feature diagram describes the valid products (configurations) in a tree structure. Leafs represent features and serve as the variation points. Their selection or deselection leads to the specification of a desired product.

The standard variability model includes the following parent-child relations:

- **Optional**: Features can be selected or deselected.
- **Mandatory**: A selected parent node requires the selection of this mandatory feature.
- **Or-group**: A selected parent node requires the selection of at least one child feature.

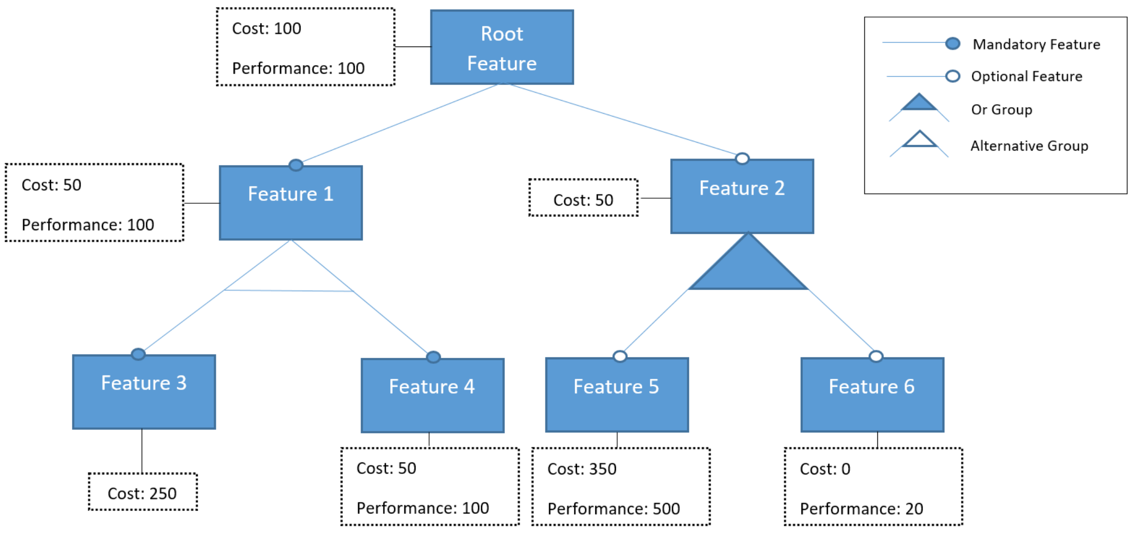


Figure 2.1: Example attributed variability model

- **Alternative-group**: A selected parent node requires the selection of at most one child feature.
- **And-group**: A selected parent node requires the selection of all child features.

Additionally cross-tree constraints define relations between configuration options outside of the parent-child relations.

- **Require**: A selected feature implies the selection of another feature.
- **Exclude**: Two features in an exclude relation cannot be selected at the same time.

Attributed Variability Models

The classic variability model only deals with the functional aspects of a configurable software system. Non-functional properties such as performance, reliability and cost cannot be modeled with it. To allow the automated reasoning on non-functional properties Benavides et al.[BTRC05] proposed an extension to variability models, the extended or attributed variability model. Attributes are annotated directly to the features and define a measured or specified influence of a single feature on a non-functional property. Every attribute belongs to a domain, which is the space of possible attribute values. The domain can be discrete (e.g. high,middle,low, true, false) or continuous (e.g. real). Figure 2.1 shows an example attributed variability model. It consists of two non-functional properties (cost, performance) in a continuous domain.

The attributed variability model enables additional analysis operations on a configurable software system. Examples of these operations are:

- Optimization: Searches for products which minimize or maximize an objective function. Example: Product with minimal total cost, where total cost is the sum of costs of each feature.

- Filter: Searches for products which are coherent to a certain requirement or limitation. Example: All products which costs are below a certain value.

2.2 Interactions

Some feature combinations influence the non-functional property of a product unexpectedly. The combination of two or more features leads to an unexpected change in the estimated non-functional property, whereas the individual selection of the features does not. These interactions between features can be found for almost every non-functional property in most real-world systems.[SKK+12]
The order of interaction specifies how many individual features take part in an interaction. A first order interaction consists of two features, a second order interaction of three features and so on.
For example: A database system has the features compression and encryption. If both are enabled the data is first compressed and then enciphered resulting in smaller data chunks to encrypt, which in turn can lead to a faster data encryption.
Interactions can usually not be detected during at the configuration phase, as the

2.3 Kernel Density Estimation

The input distributions for our tool can be of any size. In order to generate attributed variability models of the desired sizes with properties similar to real-world systems. Therefore it is necessary to resample the input distributions of feature-, interaction-, and variant values from real-world systems to any desired size. The underlying distributions of the values are however unknown.
Kernel density estimation is a nonparametric (i.e. we assume the data is not drawn from a given probability distribution) technique to estimate the probability density function of a finite dataset. Once we calculated the density function, we can draw random samples of any size from the calculated density estimation.
Let $X_1, X_2, ..., X_n$ be a sample of size n from a random variable with density f. Then the kernel density estimate of f at the point x is as described by Sheather [She04] given with:

$$\widehat{f}_h(x) = \frac{1}{nh}\sum_{i=1}^{n} K(\frac{x - X_i}{h}) \tag{2.1}$$

where the kernel K satisfies $\int K(x)dx = 1$ and h is the window width and often called the bandwidth.
In the tool's implementation the Gaussian kernel is used:

$$K(y) = \frac{1}{\sqrt{2\pi}} exp(-\frac{y^2}{2}). \tag{2.2}$$

The selection of the bandwidth parameter is crucial as it can overfit or underfit the sample. An example is shown in Figure 2.2. We show the histogram and four different kernel density estimation plots for the binary size feature values of the Berkeley DB, with sample size 8. The bandwidth parameter heavily influences the smoothness of density estimates. A false selection of the bandwidth parameter can lead to

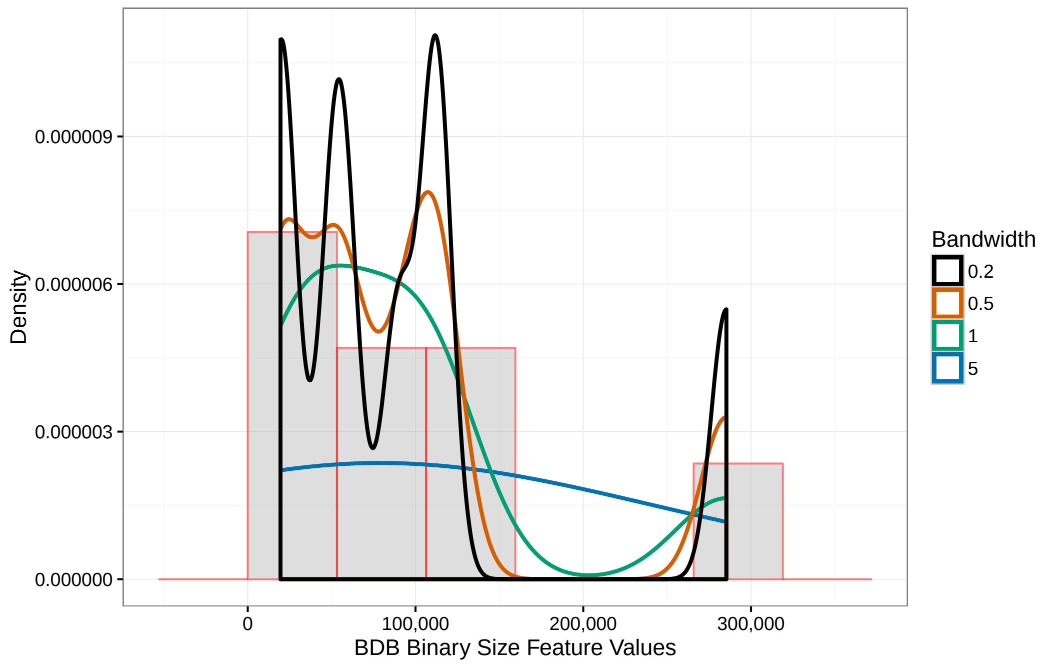


Figure 2.2: Histogram and kernel density estimation with different bandwidths

strong deviations from the original sample, especially in case of small input sample sizes.

To compute the optimal bandwidth for each data sample the Kernel Smoothing [1] package for the R language is used. It minimizes the asymptotic mean integrated square error (AMISE) with the bandwidth selector proposed by Wand et al. [WJ94] in order to receive close to optimal bandwidth parameter selections.

2.4 Pareto Optimality

In multi-objective optimization pareto optimality describes the problem of finding optimal solutions under multiple conflicting objectives, so that there does not exist a solution that is better in one objective without being worse in another objective. Formally, a solution s_1 from the solution set S of the multi-objective problem P with the objective value f_i of the ith objective (pareto) dominates another solution s_2, if:

1. $f_i(s_1) \leq f_i(s_2)$, for all objectives i, and
2. $f_j(s_1) < f_j(s_2)$, for at least one objective j

A solution is pareto optimal, if it is not dominated by another solution. The set of pareto optimal solutions is called the pareto front. Figure 2.3 shows an example pareto-front of a two-dimensional minimization problem. The blue line represents the pareto-optimal solutions which dominate the orange solutions.

[1] https://cran.r-project.org/web/packages/ks/ks.pdf

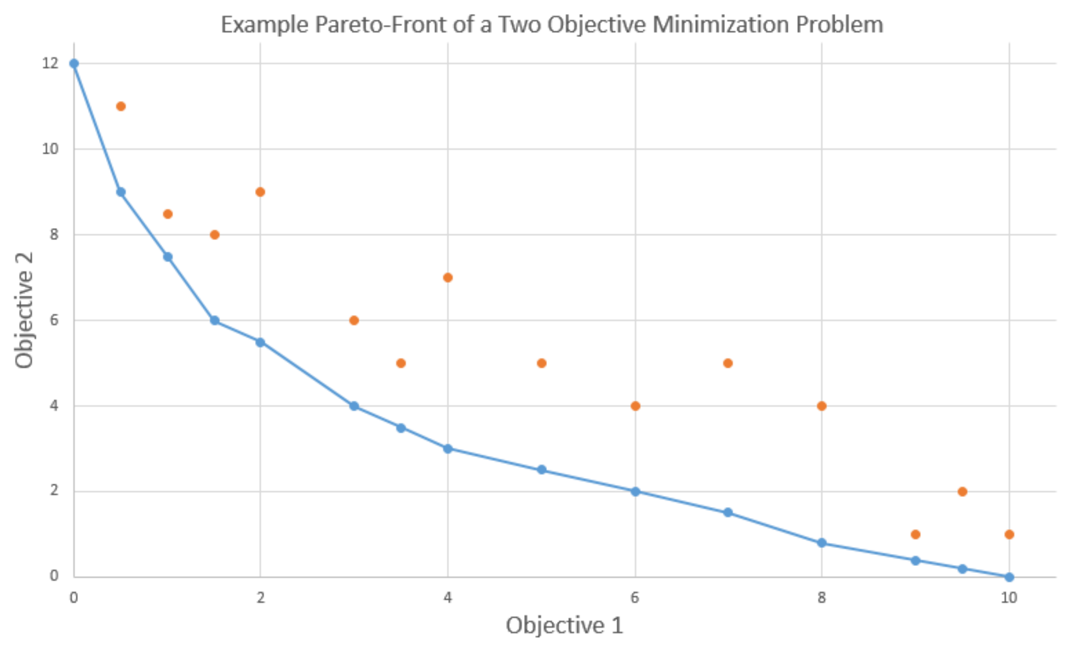


Figure 2.3: Example pareto-front of a two-objective minimization problem

2.5 Evolutionary Algorithm

Evolutionary Algorithms (EAs) are heavily inspired by biology and Darwin's evolution theory. A population of individuals is affected by the environmental pressure which forces a natural selection process - the survival of the fittest - upon them which causes a rise in the population's fitness over time. The fitness specifies how well an individual is able to survive in the current environment. Given a fitness function it is possible to evaluate a candidates quality under a certain problem encoding. Based on this fitness, a proportion of the better candidates are selected for seeding the next generation by applying a recombination or mutation operator on this selection. Performing recombination and mutation creates a new set of candidates (the offspring) that compete with the existing population for a spot in the following generation. This process starts with an initially randomly selected population and identifies a set of candidates with sufficient quality (through several iterations). The recombination and mutation operators create the diversity and promote new candidates, whereas the selection operator advances the overall quality of solution candidates over the generations.
Based on this theory evolutionary algorithms follow a fixed scheme which generally consist of the following steps. This process is also illustrated in Figure 2.4:

1. Create an initially random population of individuals.
2. For each individual calculate a objective value.
3. Assign a fitness value from the objective values to each individual.
4. Perform a selection ("survival of the fittest") where the best individuals are placed into the mating pool.

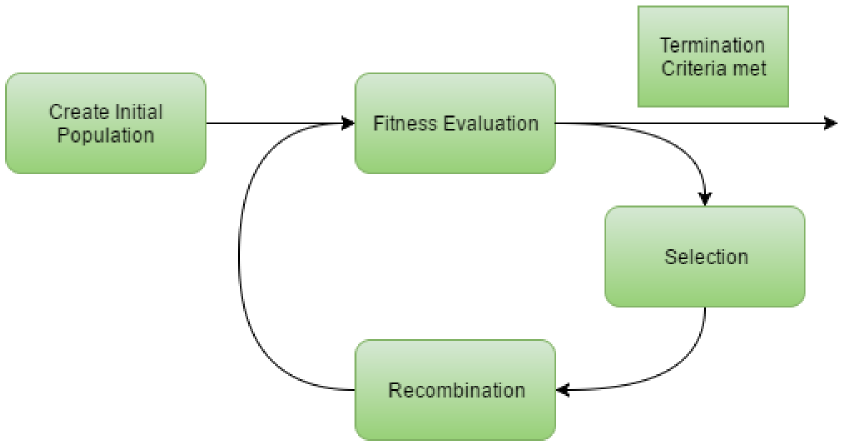


Figure 2.4: Basic cycle of evolutionary algorithms

5. A recombination step performs crossover and mutation on the mating pool. Those new individuals are integrated into the population.
6. Lastly a survivor selection is performed to eliminate candidates with poor fitness.
7. If a termination criteria (e.g. solutions sufficiently good enough, or maximal amount of iterations met) is reached then stop. Otherwise continue with step 2.

The following sections describe the components of evolutionary algorithms.

2.5.1 Problem Encoding

The problem encoding is essential to the applicability of an evolutionary algorithm to an optimization problem. It provides a mapping from the physical representation of a candidate to the qualities of an individual in the problem to be optimized by an evolutionary algorithm. The similarity to genetics is characterized by the complex mapping between a genotype (the collection of genes possessed by an individual) to the properties of an organism, called the phenotype. The structure containing the collection of genes of an genotype is referred to as the chromosome.[DM97]
The choice of encoding varies greatly with the chosen optimization problem and certain problems might only be solvable with the correct encoding type. The most common encodings are explained by examples:

Binary encoding In binary encoding every chromosome is a Boolean array, where true (encoded as 1) represents the presence of a gene in an chromosome.

Chromosome A	1001010101010101001
Chromosome B	0010100011011011 0

Example: A knapsack has a given capacity and there are items with given size and value. Select the most value of items without exceeding the capacity.
Encoding: Each item of the knapsack represents a gene.

Permutation encoding Each chromosome is an array of integers, which represent the number in a sequence.

Chromosome A	3 6 2 1 7 4 5
Chromosome B	7 1 5 2 4 3 6

Example: A traveling salesman has to visit all cities which have a given distance between them. Find the minimal travel distance.
Encoding: The chromosomes defines the order of cities, in which the salesman will visit them.

Value encoding Every chromosome is an array of values, where the value can be of any type related to the problem (e.g. real, complex, other objects).

Chromosome A	1.75 2.23 8.10 3.14
Chromosome B	0.71 5.33 2.60 4.88

Example: This encoding is also used in our generator tool.
Encoding: Each gene represents an attribute value of a feature.

2.5.2 Fitness Function

The biological fitness defines the ability of an individual to survive and reproduce in the current environment. Fitness functions of evolutionary algorithms provide a mapping from the composition of genes to the quality of a candidate. Being the foundation of the selection process, it facilitates the populations increase in quality towards the solution over the generations.
The fitness calculation process generally consists of two steps:

- Calculate an objective value for each objective and individual using an objective function in compliance to the problem encoding. The objective function provides a mapping from the individual's genetic properties to a scalar value (e.g. real). There can exist different objective functions for each objective.
- From the set of objective values of an individual calculate the fitness of the candidate. The fitness function creates a total ordering and assigns a rank for each individual based on its objective values. The fitness calculation may not solely rely on the objective values, but can also include information about the whole population, such as the distance of an individual to the neighboring solution candidate, or the niching factor/crowding distance (sum of distances to near individuals). There exist different methods for the creation of the ranking of solution candidates, such as the weighted sum ranking, pareto-ranking, or the variety preserving ranking which includes the niching information into the ranking.

The design of the fitness function is next to the problem encoding the most important aspect of an successful evolutionary algorithm, as it reflects the target function or target state of the whole optimization process. Furthermore the fitness calculation needs to be done in every evolution step for each candidate in the population. This encourages the usage of computationally inexpensive operators to reduce the total amount of used resources.

2.5.3 Selection

The selection process enforces the survival of the fittest by choosing candidates for reproduction based on their fitness value. The fitter individuals are more likely to be included in the mating pool for the next generation. If an individual is selected for reproduction it is called a parent. Selection operators can be deterministic or randomized and might also incorporate an archive of previously found best individuals called the elites. Evolutionary algorithms that carry over a proportion of best individuals to the next generation are called elitist EAs, otherwise the next generation consists solely of the offspring of the parents and are referred to as generational EAs. The selection pressure is the degree over which better individuals are favored over worse ones. If the pressure is too low, the evolutionary algorithm is slower in finding optimal solutions, as more suboptimal individuals are placed into the mating pool. If however the pressure is too high, the evolutionary algorithm has an increased chance of prematurely converging to a suboptimal solution.[MG95]
The next sections describe some selection operators in detail.

2.5.3.1 Truncation Selection

The truncation selection is the simplest form of selection operators. The population of size n is sorted by their fitness value in descending order. Afterwards the k best individuals are placed into the mating pool. If $k > n$, then the best individuals are taken multiple times.

2.5.3.2 Roulette Selection

In roulette selection the population is sorted by its descending normalized fitness. Afterwards the accumulated normalized fitness value is computed for each individual, such that the best candidate has the accumulated fitness value of 1. A random number R between 0 and 1 is chosen. All individuals whose accumulated fitness value is greater than R are placed into the mating pool. This procedure is then repeated until the mating pool reaches the desired size.

2.5.3.3 Tournament Selection

A tournament is hold, where s (tournament size) random individuals compete against each other. The individual with the highest fitness value is placed into the mating pool. This is repeated until the mating pool is full. The choice of the tournament size is essential and controls the selection pressure. Small tournament sizes decrease the selection pressure, as lower valued candidates are more likely to win the tournament. High tournament sizes increase the selection pressure, as lower valued candidates are more likely to compete against better opponents. Tournament selections are therefore very popular, as it allows the adjustabillity of the selection pressure while still maintaining a certain randomness in comparison to the truncation selection.[Luk13]

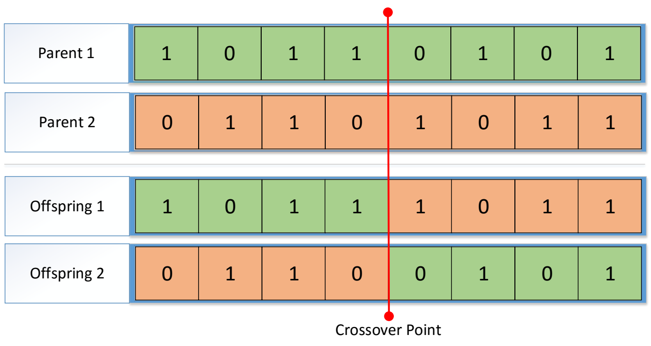


Figure 2.5: Example Single Point Crossover with Binary Encoding

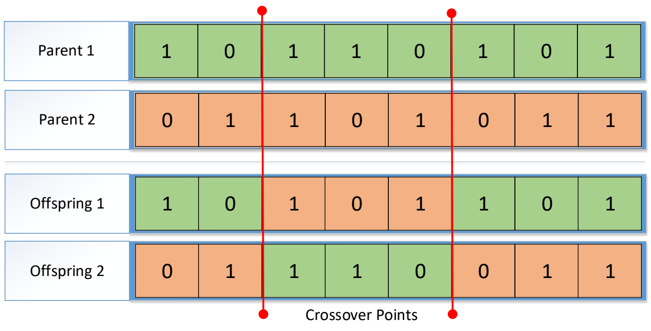


Figure 2.6: Example Two-Point Crossover with Binary Encoding

2.5.4 Crossover

The crossover operator is part of the recombination step, where new individuals are formed from two parents. It is analogue to biological reproduction where the children inherits some of the genes of each parent. Together with the mutation operator the crossover is the foundation for the discovery of new candidates. The next sections describe three different crossover operators.

2.5.4.1 One-point Crossover

A single crossover point on both parents is selected at random. The genes after this point are then swapped and result in the new offspring. Figure 2.5 visualizes the one-point crossover selection.

2.5.4.2 Two-point Crossover

Two crossover points are selected at random. The genes between those points are swapped and result in the new offsprings. An example two-point crossover is depicted in Figure 2.6.

2.5.4.3 Uniform Crossover

In uniform crossover each gene is selected with a defined probability from the first parent, otherwise the second parent is chosen.

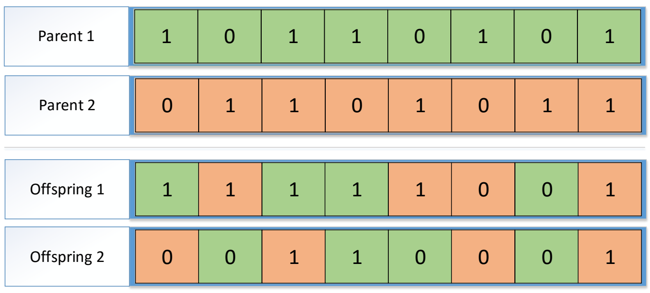


Figure 2.7: Example Uniform Crossover with Probability 0.5

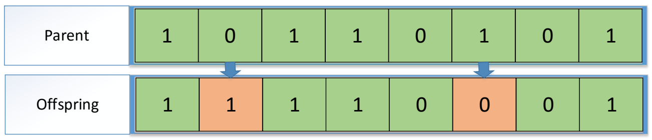


Figure 2.8: Example Bit-flip Mutation

2.5.5 Mutation

Similar to the biological mutation, the mutation operator from evolutionary algorithms changes the genes from one or more chromosomes. It is part of the recombination phase and mostly applied after the crossover operator. It is the component of the genetic algorithm which counters the premature convergence to suboptimal solutions. The kinds of applicable mutations depend on the problem encoding.

2.5.5.1 Bit-flip Mutation

Bit-flip mutation selects a gene with a probability of $\frac{1}{n}$, where n is the length of the chromosome. The binary value of that gene is inverted. On average only a single mutation appears in one step. The bit-flip mutation is only applicable to binary encoded problems.

2.5.5.2 Uniform Mutation

Uniform mutation is a mutation operator for value encoded problems. The probability of a gene's mutation is $\frac{1}{n}$, where n is the length of the chromosome. The new value is selected uniformly from a lower and upper bound, which needs to be set according to the boundaries of the value encoding.

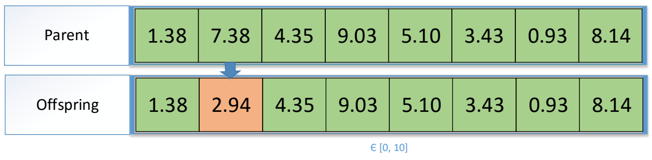


Figure 2.9: Example Uniform Mutation

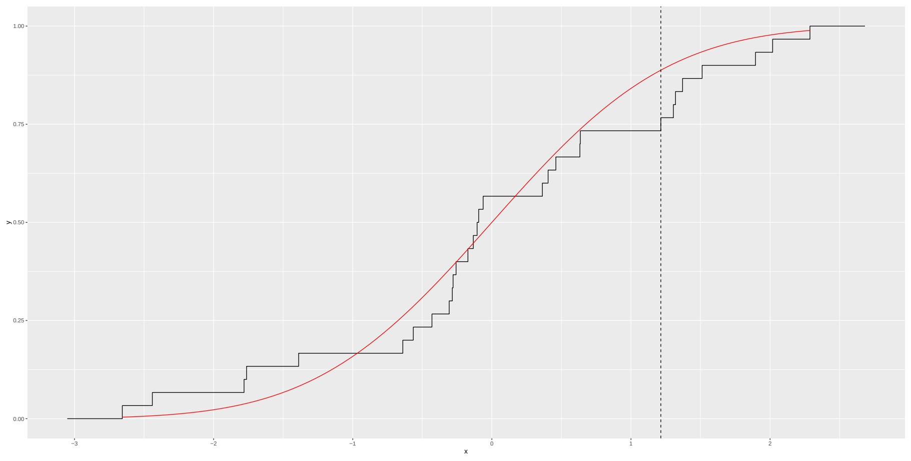


Figure 2.10: Example KS-Test: Dotted line shows the supremum of the distance between the empirical distribution function ad a probability distribution (red)

2.6 Statistical Tests

To tune our generated attributes close to the desired real-world attribute distributions, it is necessary to determine the grade of similarity, or goodness-of-fit, between the targeted real-world distribution and the current distribution of the genetic algorithm's population. We implemented four different techniques for the comparison of distributions, two statistical tests and two binned distance metrics.

2.6.1 Kolmogorov-Smirnov Test

The Kolmogorov-Smirnov test (KS) is a nonparametric (e.g. it is not dependent on a parameter) test used to check the probability whether a sample distribution comes from a specified probability function (one-sample KS-test) or if two samples originate from the same distribution (two-sample KS test). We can use this test to determine the goodness-of-fit of two sample distributions.
The test-statistic of the KS test is specified as the maximum distance between the empirical distribution function and the given probability function or, in the two-sample case, the maximum distance between both empirical distribution functions. Formally, for $(x_1, ..., x_n)$ independent, identically distributed real random variables with the common cumulative distribution function F(t), the empirical distribution function is defined as:

$$\widehat{F}_n(x) = \frac{1}{n}\sum_{i=1}^{n} 1_{x_i \leq t} \tag{2.3}$$

The Kolmogorov-Smirnov statistic for a empirical distribution function $\widehat{F}_n(x)$ and a probability function $F(x)$ is then defined as:

$$D_n = \underset{x}{sup}\left|\widehat{F}_n(x) - F(x)\right| \tag{2.4}$$

The one-sample test can be used to compare a sample with a reference probability distribution, which is not useful for our purposes since we do not want to test the resemblance to normal or uniform distributions, but rather compare two distribution's goodness-of-fit. Therefore we use the two sample KS test, whose output is the maximum distance of two empirical distribution functions.

$$D_{n,n'} = \sup_{x} |F_{1,n}(x) - F_{2,n}(x)| \tag{2.5}$$

where $F_{1,n}$ and $F_{2,n}$ are the empirical distribution functions of both samples.

2.6.2 Cramér-von Mises Test

The CmV test is a rank-based distance metric for the goodness-of-fit of a cumulative distribution function of a sample and a given empirical distribution function. Like the Kolmogorov-Smirnov test it can also be used to calculate the goodness-of-fit of two empirical distributions.
To compute the two sample Cramér-von Mises test statistic following steps are necessary:

1. Sort both samples in ascending order.
2. Assign a rank to each value of the combined sample.
3. Sum the squared distances between the index and the rank for each value times the sample size according to equation 2.7. Figure 2.11 illustrates this step.
4. Calculate the test-statistic T as shown in equation 2.6

The equations as described by Anderson[And62]:
Let $x_1, x_2, ..., x_N$ and $y_1, y_2, ..., y_M$ be the first and second sample with size N and M in ascending order. Let $r_1, r_2, ..., r_N$ be the ranks of the first sample, and $s_1, s_2, ..., s_M$ be the ranks of second sample in the combined sample. Then the test statistic T is defined as:

$$T = \frac{U}{NM(N+M)} - \frac{4MN-1}{6(M+N)} \tag{2.6}$$

where

$$U = N\sum_{i=1}^{N}(r_i - i)^2 + M\sum_{j=1}^{M}(s_j - j)^2 \tag{2.7}$$

The CmV test is the only test we implemented that uses a rank-based comparison method. Rank-based techniques can be problematic as they do not operate on the actual values of the sample. Consider the sample [1,2,3,4,5]. We want to compare it with the sample [3,4,5,6,7] and also with [3,4,5,99,100]. Clearly the two samples differ largely at their tails, but in both cases the two samples will receive the same ranks and consequently the same test statistic. Therefore we expect the CmV test to perform poorly if there are large gaps in the samples under test.

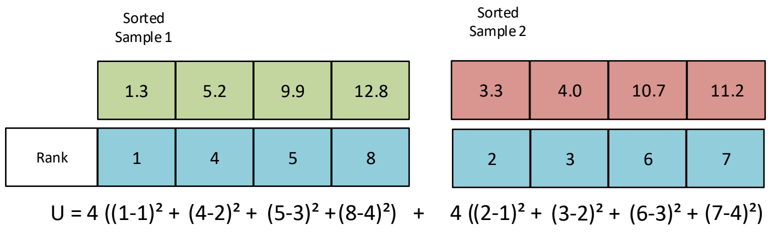

Sorted Sample 1
Sorted Sample 2
1.3
5.2
9.9
12.8
3.3
4.0
10.7
11.2
Rank
1
4
5
8
2
3
6
7
U = 4 ((1-1)² + (4-2)² + (5-3)² +(8-4)²) + 4 ((2-1)² + (3-2)² + (6-3)² + (7-4)²)

Figure 2.11: Ranked-based Distance Calculation in Cramér-von Mises Test

2.6.3 Binned Distance

Binned distance metrics operate on histogram representations of the samples. The value range of the sample is divided into bins of equal length, which are then filled with the values falling into this range. The distance is then calculated on the bin values of the sample and not on the actual values. Hence the test statistic depends on the selection of the bin count and the resulting bin width. Both samples need to be ordered into an histogram bin collection of the same size.
Two binned distance metrics are implemented in the tool, which are presented in the following paragraphs.

Euclidean Distance

Let $x = x_1, x_2, ..., x_n$ and $y = y_1, y_2, ..., y_n$ be the bin counts of two histograms with size n, then the Euclidean distance is defined as:

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{2.8}$$

Chi-Squared Distance

Let $x = x_1, x_2, ..., x_n$ and $y = y_1, y_2, ..., y_n$ be two histograms of size n, then the Chi-squared distance is defined as:

$$d(x, y) = 0.5 \sum_{i=1}^{n} \frac{(x_i - y_i)^2}{x_i + y_i} \tag{2.9}$$

3. Literature study

This chapter is divided into two parts. At first we present the key findings of our literature study which investigated the usage of attributed variability models, initial generation of attribute values, and the presence of feature interactions in current research. The second part visualizes the distributions from the SPLConqueror dataset, in order to spot patterns in real-world attribute distributions. These insights are then utilized for the further improvement of the tools test statistics.

3.1 Literature Review

We performed a literature survey to assess the current state of the art of attributed variability. In a selection of 90 studies on attributed variability we investigated the following issues:

- Research domain: Describes which areas of research are covered and which problems with attributed variability are solved.
- Sources of attributed variability models: Outlines how attributed variability models are obtained in current research.
- Generation of attribute values: Characterizes the attribute value distributions used in research and their initial generation.
- Recognition of Feature Interactions: Inspects to what extend feature interactions are handled in current research.

3.1.1 Research Domain

Non-functional properties are the focus of a variety of different research topics in the software product line area. They cover different topics, from optimization of configurations under global constraints, over test case creation with minimal costs and maximal utility, to estimation or prediction of performance values, and many other applications. Although our selection of papers on attributed variability covers a wide spectrum of different problems and solution approaches, it is possible to broadly classify the surveyed works into five categories.

- Optimal feature selection: Consists of studies dealing with the search for one optimal configuration - or pareto-fronts of optimal configurations under one or more conflicting constraints. The search can be conducted either at configuration time or dynamically at runtime. Results of the search can be either approximative or exact.
- Non-functional property prediction: Stands for works on different heuristics for measurements and model-based methods to predict the non-functional properties of configurations and features.
- Variability modeling: Is composed of topics on the modeling of quality attributes. Domain-specific languages as well as modeling approaches like custom ontologies belong in this class.
- Variability model analysis: Contains publications which focus on the analysis of attributed variability. Defines methods and implementations of operations specialized on attributed feature models.
- Other: All publications which did not fall in one of the previous categories. These consist of other literature surveys, parallelization techniques for solvers, the synthesis of attributed feature models from product descriptions or the modular verification of software product lines only to name some approaches.

As Section 3.1.1 illustrates, optimal feature selection is the most dominant category with 48 out of 90 case studies. The next biggest category consists of 14 papers on the prediction or estimation of non-functional properties. Lastly, modeling of attributed variability and literature surveys on attributed variability appear in 4 papers each.

We also analyzed the approaches which were used in studies on the optimal feature selection problem. The results can be seen in Section 3.1.1. By far the most popular method are evolutionary algorithms. Almost one third of publications used a genetic algorithm for the optimal feature selection problem. 7 papers chose a constraint-satisfaction-problem solver which provides exact solutions in comparison to the evolutionary algorithm.

3.1.2 Sources of Attributed Variability Models

The variability models we found in the surveyed papers mainly came from one of these three sources:

- Generated by a feature model generator such as BeTTy[1]

[1]http://www.isa.us.es/betty/betty-online

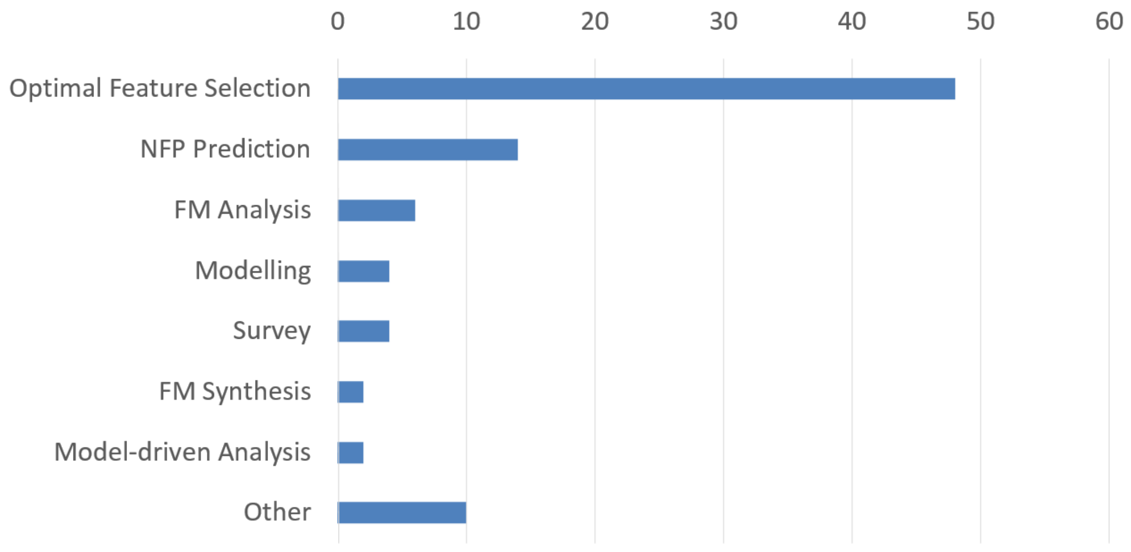


Figure 3.1: Classification of surveyed papers into research domains

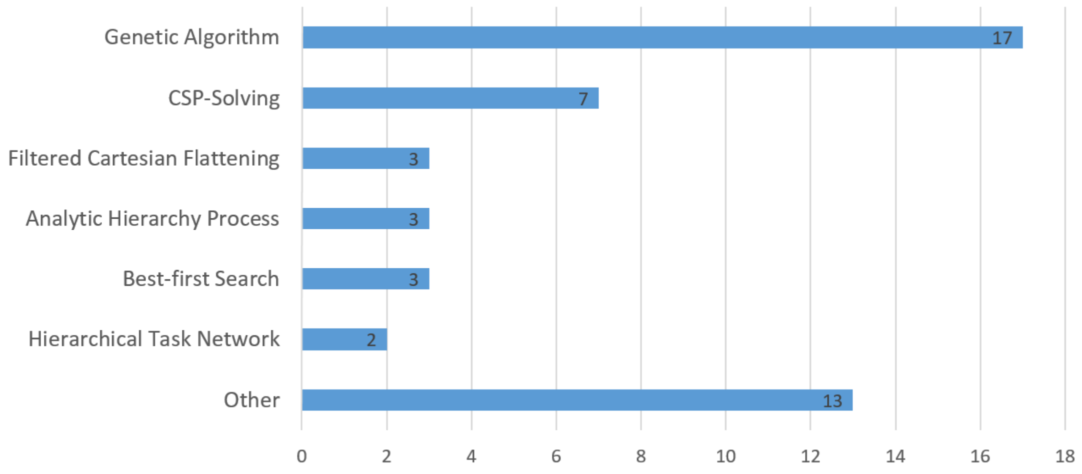


Figure 3.2: Classification of optimal feature selection approaches

- Taken from online feature model repositories such as the SPL Online Tools, SPLOT[2] or from the Linux variability analysis tools, LVAT[3].
- Created for or derived from real-world systems or academic case-studies.

The SPLOT has a large repository but their models are rather small (<366 features) compared to the Linux studies of the LVAT repository (up to 13000 features). The models found in SPLOT and LVAT contain no attributed variability. Only BeTTy allows the creation of attributed variability models to a limited extend. Each of these tools has no mention of feature interactions, although BeTTy can easily be extended to include interactions.

3.1.3 Initial Generation of Attribute Values

Real-world attributed feature models are hard to obtain, because the measurement of larger real-world systems is a time consuming process and proprietary data is often not freely available. The authors then tend to take randomly generated feature models or those from the known online repositories like SPLOT for the evaluation of their approaches.
With the exception of the BeTTy online model generator, which can generate attributes to features, these feature models are not enriched with non-functional properties. Therefore, in order to use these models in studies on attributed-variability they have to be annotated with non-functional properties, either by manual or automatic generation utilizing a random function.
This section describes how attributed values were added to the features in their respective feature models. Figure Figure 3.3 illustrates in which manner the surveyed publications added attributes to their model.
Overall 51 papers generated non-functional properties, 16 of these did not specify their method, 12 used a uniform distribution, 12 added the attribute values manually, 9 defined multiple attributes and used normal or uniform distributions and lastly 2 works used a normal distribution.
In total 18 papers used measurements from existing systems for their attribute values. With 12 publications, the dataset from the SPLConqueror tool forms the majority of used measurement values. Only 4 publications conducted their own measurements, and 2 papers did not provide information on how they performed the measurements.
The remaining publications provided no information on the source of their attribute values.
One particular frequent kind of attribute generation is given by Sayyad et al. [SA13, SGPMA13, SMA13, SIMA13a] who defines three attributes:

- Cost: Describes the cost of a feature. Normal distribution, real values in the range from 0.0 to 15.0
- Defects: The amount of found errors. Normal distribution, integer values from 0 to 10

[2]http://www.splot-research.org/
[3]https://code.google.com/archive/p/linux-variability-analysis-tools/

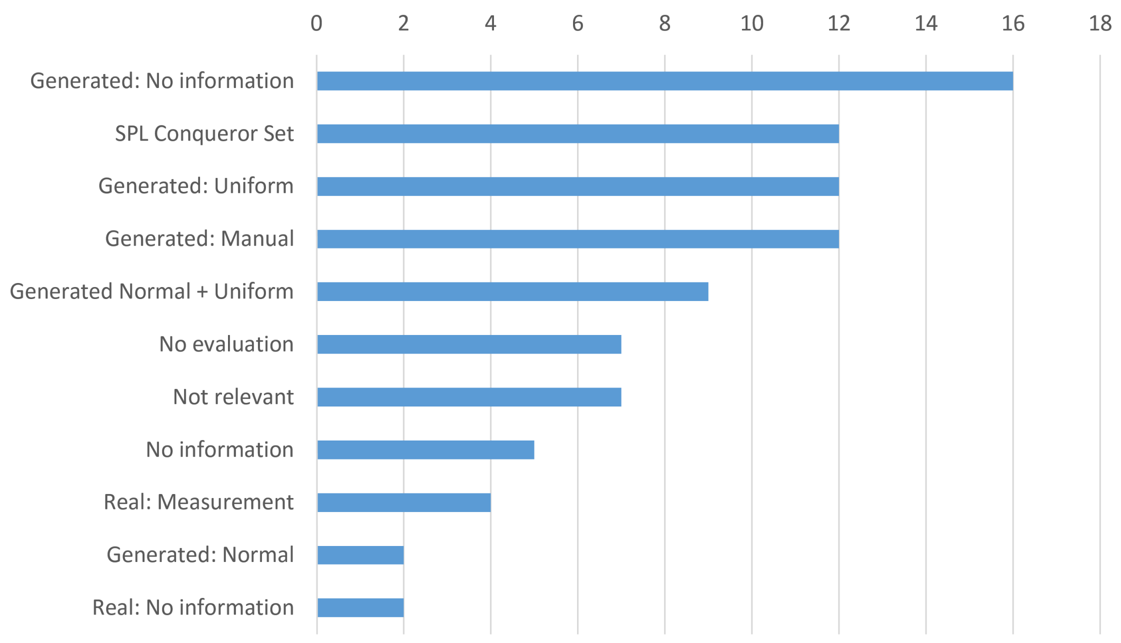


Figure 3.3: Counts of the found attribute generation methods

- Used_before: Indicating if a feature was used in a previous configuration. Uniform distribution, boolean values

In order to compare their approaches many authors then adopted this generation method, even if Sayyad explicitly stated the danger of synthetically generated attributes in the threats to validity section. These among others include Tan [TXC$^+$15], Lian [LZ15], Zhang [ZYL14], Olaechea [ORGC14] and Henard [HPHLT15]. The drawback of randomly generated distributions is the inadequate similarity to non-functional property values of real-world systems. We did not find a single system in the SPLConqueror dataset where the attribute distributions are equal or close to a normal or uniform distribution. Figure Figure 3.4 shows the histograms and kernel density estimation of the main memory feature values of SQLite and the LLVM performance feature values on the left side. The right side shows a sample uniform and normal distribution as used by many authors in the surveyed studies. It can be seen that real-world systems do not follow ideal distributions like the generated ones and often contain strong peaks, gaps, and statistical outliers in the density function.

3.1.4 Feature Interactions

A feature interaction is present when two or more selected features in a configuration lead to an unexpected change in a non-functional property, but their individual selection does not.
From analyses of real-world systems including variability it can be seen that actually every configurable system contains interactions [SKK$^+$12]. From the visualization of the SPLConqueror dataset we know that many of these feature interactions have rather small impact on the non-functional properties, but their quantity () and few single outliers who greatly influence a systems non-functional property, provide an

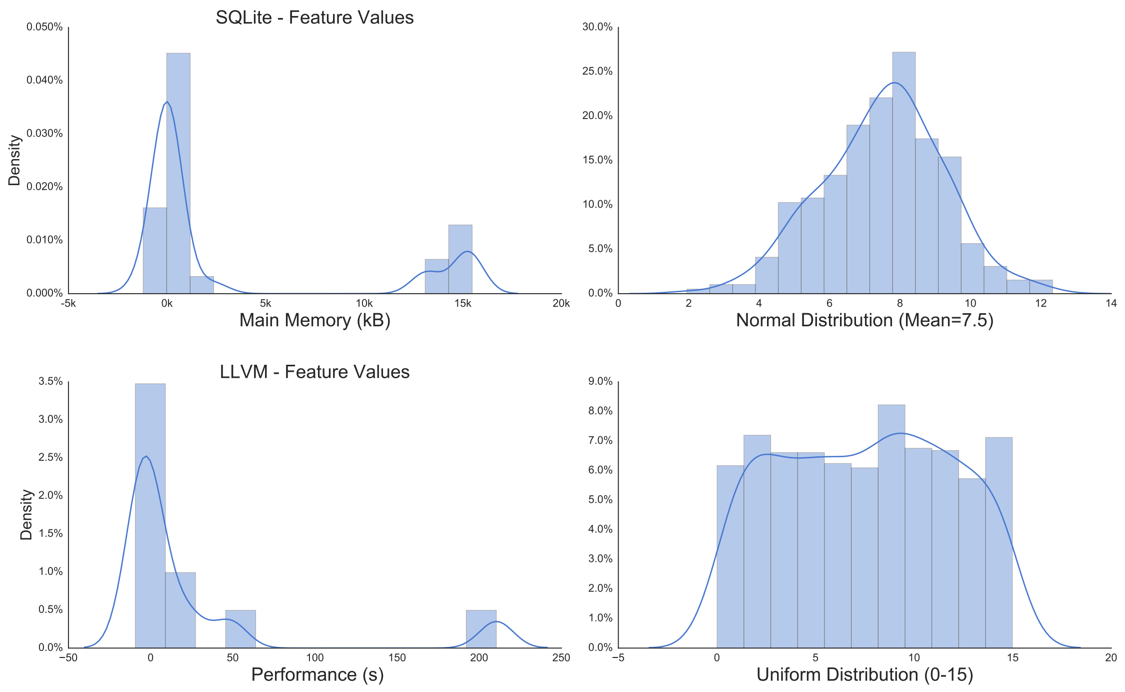


Figure 3.4: Measured Main Memory (SQLite) and Performance (LLVM) distribution profiles compared to Normal and Uniform Distributions

undeniable impact of feature-interactions on non-functional properties of a configurable system.

Figure Figure 3.5 shows the measured performance values of the LLVM compiler

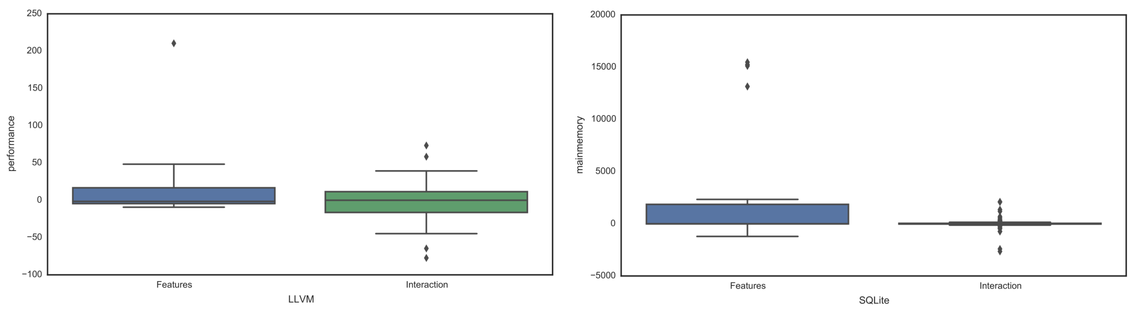


Figure 3.5: Feature values and interaction values of LLVM performance and SQLite memory consumption in comparison

system and the memory consumption of SQLite for features and the feature interactions. In both cases it can be seen that many feature interactions have close to zero impact on their respective non-functional property, but outliers exist whose impact can not be neglected. The amount of interactions, for n features, there a 2^n possible feature-interactions, also increases the impact of feature-interactions. In the SPLConqueror dataset, we noticed that the amount of feature interactions is often in the same magnitude as the amount of features, and more feature-interactions are present the more features the configurable system has. From the surveyed papers only 10 publications from 6 authors handle interactions. 5 of them mention feature

Binary Size	Feature Values	Interaction Values	Variant Values
BerkeleyDB	✓	✓	✓
EPL			✓
LinkedList	✓	✓	✓
Linux	✓	✓	
PKJab	✓	✓	✓
Prevayler	✓	✓	✓
SNW	✓	✓	
SQLite	✓	✓	✓
Violet	✓	✓	✓
ZipMe	✓	✓	✓

Table 3.1: Available Binary Size Measurements

interactions in their evaluation or threads to validity section, but do not further include them in their approaches. The majority (75) of publications do not include any kind of feature interaction nor discuss their absence in the evaluation, threats or future work sections.
Figure Figure 3.6 shows the repudiation of feature interactions graphically.

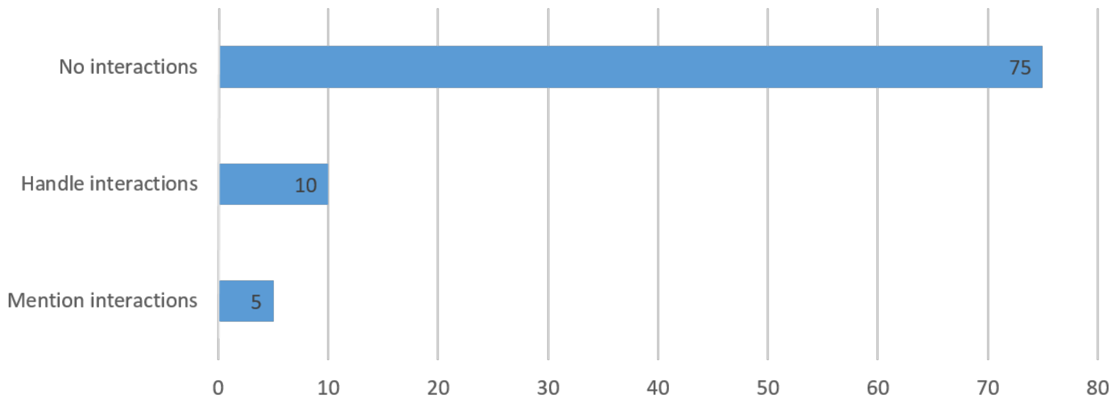


Figure 3.6: Amount of publications with handled, mentioned, and ignored interactions of the literature survey

3.2 SPLConqueror Dataset

This section visualizes the SPLConqueror dataset. It illustrates which data from the configurable systems under test are available in the dataset. Measurements were performed for the non-functional properties main-memory, performance, and binary size. The tables Table 3.1, Table 3.3, Table 3.2 describe which data is available for each of the measured properties binary size, performance, and main memory. With the exception of some systems the datasets are exhaustive for features, interactions, and system variant values.

Performance	Feature Values	Interaction Values	Variant Values
AJStats	✓	✓	✓
Apache			✓
BerkeleyDBC	✓	✓	✓
BerkeleyDBJ	✓	✓	✓
Elevator	✓		✓
Email	✓	✓	✓
LLVM	✓	✓	✓
x264	✓	✓	✓
ZipMe	✓	✓	

Table 3.2: Available Performance Measurements

Main Memory	Feature Values	Interaction Values	Variant Values
BerkeleyDB	✓	✓	✓
Curl	✓	✓	
LLVM	✓	✓	✓
SQLite	✓	✓	✓
Wget	✓	✓	✓
x264	✓	✓	✓

Table 3.3: Available Main Memory Measurements

We created histograms and density plots for each of the dataset in order to spot similarities and differences in the datasets and to recognize patterns which can provide useful insights for the tools implementation. We broadly classify the attribute distributions from the dataset into the following categories:

- **Single peak:** Describes distributions with a single centered maximum and very few deviances at the sides of the center. Examples of these can be found in Figure 3.7.
- **Skewed peak:** Similar to the single peak distributions, only with the maximum shifted to one side of the distribution. Figure 3.8 shows some sample distributions.
- **Multiple peaks:** Multiple maximums, with large gaps between single peaks. Often one maximum is greatly larger than the others. This category is the most frequent with around 60 % of all distributions which have two or more peaks. The examples for this category are shown in Figure 3.9.
- **Needle in a Haystack:** Strong multiple peaks with attribute values clustered around a peak, and with mostly smaller outliers at a greater distance. This is the second most frequent distribution pattern with around 27%. Needle distribution patterns are illustrated in Figure 3.10.

We expect this order of categories to increase in difficulty for the genetic algorithm, as the distributions differ more and more from the genetic algorithms start with the normal distribution. For evaluation results on these types of distributions we refer to the experiments and evaluation chapters. The complete set of plots can be found in the appendix.

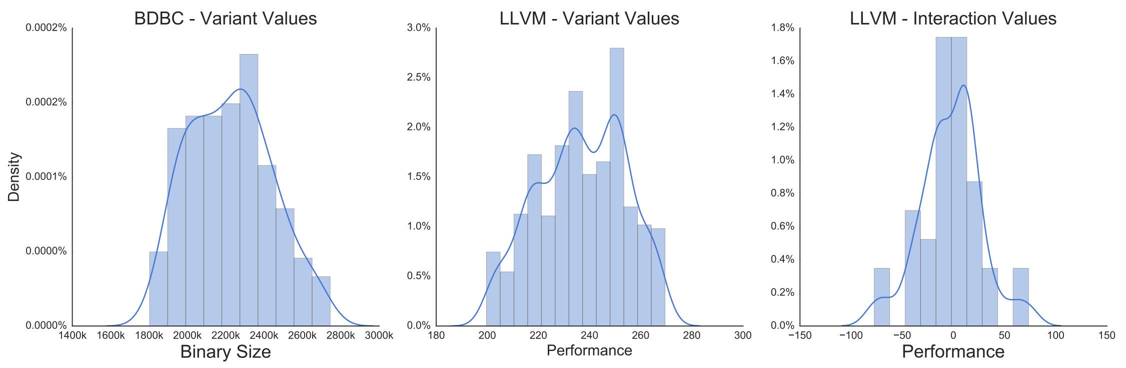


Figure 3.7: Examples of single peak distributions

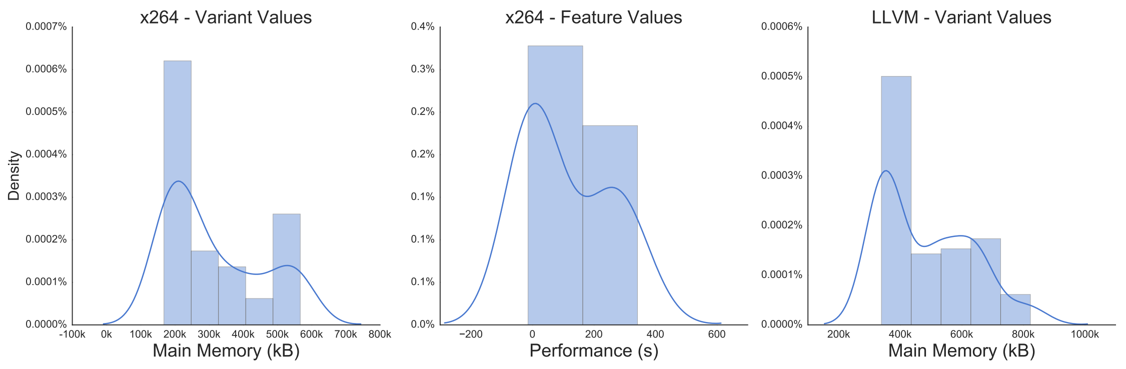


Figure 3.8: Examples of skewed single peak distributions

3.3 Results

This section summarizes the results from the literature study and the SPLConqueror dataset analysis with regard to the research questions proposed in the introduction. Concerning **LS RQ1**, from the selection of 90 surveyed publications, 69 required attributed variability models in their approaches. From these only the minority of publications (18) are using measured attribute values, with 12 studies reusing the existing SPLConqueror dataset. Only 6 works, measured their own attribute values or obtained measurements from somewhere else.

The remaining publications (51) are using artificially generated attribute values. Close to three quarters of publications use synthetic attribute value distributions such as normal, or uniform distributions. Unfortunately most of the authors (16) did not explicitly state which distributions were used. In case of multi-objective problems multiple distributions were used on different attributes.

The literature study also answered **LS RQ2**. Only 10 research papers are including feature-interactions in their approaches. Just 5 are at least mentioning the presence of interactions in their evaluations or threats to validity section, but do not actively

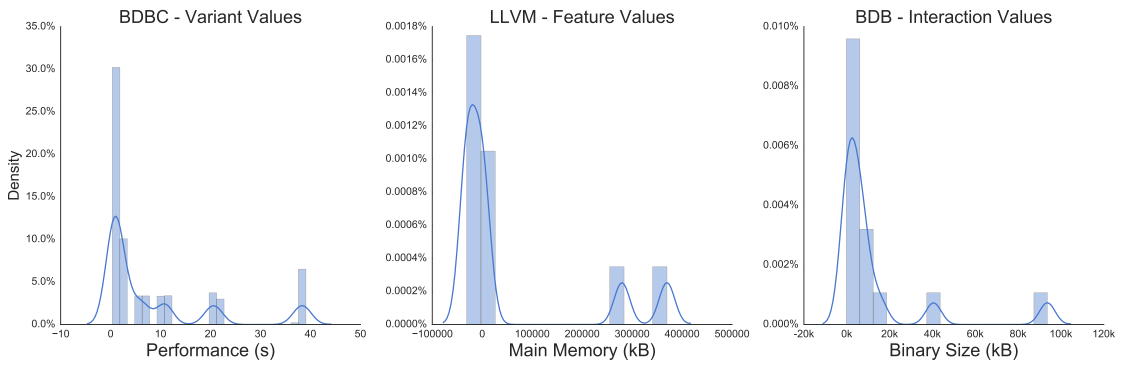


Figure 3.9: Examples of distributions with multiple peaks

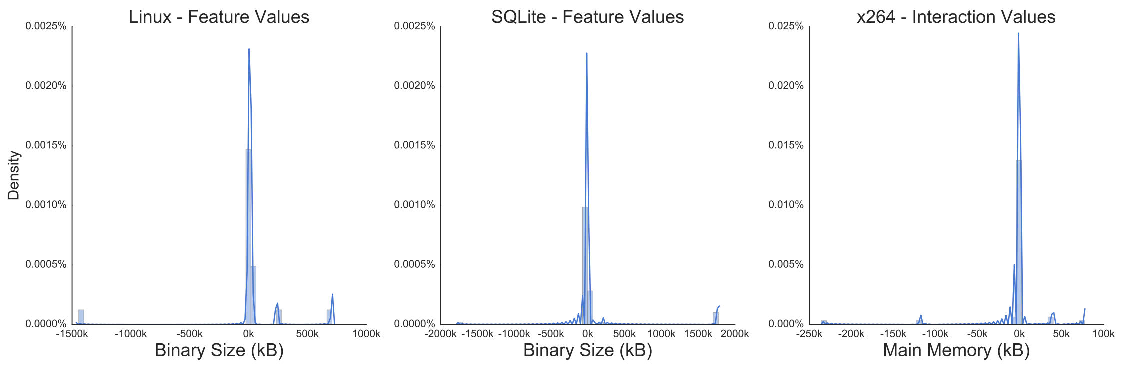


Figure 3.10: Examples of "Needle in the haystack" distributions

handle them. The vast majority (75) neither handle or mention interactions in any way.
The dataset analysis, for **DA RQ1**, revealed four general patterns in the attribute value distributions, whereas the transitions from one class to another are crossing over such that is possible to assign some profiles to both classes. The multiple-peak distribution pattern appeared the most frequent with 60% of all distributions in the SPLConqueror dataset. Followed by the needle distribution pattern, with 27% of distributions. Single peak and skewed peak are equally rare with around 6% of all distributions.
We did not find a single distribution of attribute values from real-world configurable systems in the SPLConqueror dataset which is resemblant to a normal or uniform distribution.

4. Problem Statement

The purpose of this thesis is the creation of attributed variability models including feature-interactions, where the models attribute values for the features, interactions, and system variants are similar to targeted real-world distribution profiles. One example application could be: Generate an attributed variability model with 500 features and 200 feature interactions, whose feature attribute distribution is similar to the BerkeleyDB performance distribution, the interaction distribution similar to the x264 performance distribution, and the attribute distribution over all variants is like the one from the Linux kernel. Because the attribute values for the features and interactions determine the attribute values of each variant one cannot simply set the attribute values to the targeted real-world distribution: The characteristics of the feature model, such as the size, constraints, and cross-tree constraints will lead to different variant attribute distributions for different feature models. Therefore, to achieve a high similarity of feature, interaction, and variant distributions, we need an optimization process to explore different competing assignments of attributes to features and interactions, to find one assignment that suffices the requirement of similarity to a targeted variant distribution.
Because we initially have no knowledge of the influence of feature and interaction attribute assignments to a variants distribution, we chose the genetic algorithm as the optimization technique. It enables the learning from random assignments out of the large search space to converge to an optimal solution. The involved randomness of genetic algorithms also helps to find uncommon solutions.
For the generation of attributed variability models of arbitrary size with genetic algorithms, several problems have to be addressed which will be described in the following sections.

4.1 Bootstrapping and Data Scaling

Currently the only dataset for realistic attribute values comes from the SPLConqueror dataset. We utilize these as the selectable target distributions for features, interactions and system variants. But the number of available attribute values for each system is rather small (<100) compared to the large variability models we want

to be able to generate, as for example sizes equivalent to the Linux kernel (>10000). This is problematic because we need to assign an attribute value to each feature and interaction. It is not possible to simply assign the values multiple times to features and interactions, because it would be unrealistic that multiple features have exactly the same influence on a non-functional property. Therefore, we need a method to capture the underlying structure, the probability distribution, of the selected target distribution and be able to transfer it to arbitrary sizes. We want to pull random samples from the underlying structure of the target distribution, which are similar in their characteristics to the source distribution, in order to fill the missing values for larger models. The kernel-density estimation appears to be a good candidate for this type of problem.
The kernel-density estimation is also problematic for some realistic distributions which consist only of very few values (e.g. BerkeleyDB binary size feature attribute consist of only 8 values) because the precision of the estimation increases with the amount of source values. This makes it necessary to test, if a drawn large random sample from a probability distribution estimate of such small input values is still similar "enough" to the initial sample. This requires a goodness-of-fit test between the input sample and the drawn random sample of arbitrary size.
Lastly we want to be able to select feature, interaction and variant distributions from different systems and combine them into the new attributed variability model. Different systems have different value ranges in their attribute data, for which we need to take countermeasures. For example, we should not take feature values in the range [0,10] and weave in interaction values in the range [50000-100000]. This is also the case when we increase the size of the variability model. Larger variability model will likely lead to configurations with more features and consequently will the variant distribution be in another value range as the original measured variant values. To make the distributions comparable we must adjust the value ranges and rescale the attribute values carefully when needed.

4.2 Variant Generation Sampling

For the calculation of the attribute distribution over the variants it is necessary to generate configurations of the variability models. This is infeasible for the desired size of variability models due to the variant explosion and the resulting computation effort. We must therefore reduce the amount of generated variants to a subset of all possible configurations.
The amount of variants on the one hand influences the accuracy of the distribution of the attributes over all variants. On the other hand, it is bounded by the available time, memory and processing power. Therefore, we must choose an amount of generated configurations, which is large enough, but must still be computable within the machines available resources.
Even more important than choosing an appropriate amount of generated variants is the problem of the selection of a suitable subset of variants. If the selected sample of generated variants does not correspond to the structure of all configurations of the variability model, then the resulting variant distribution will be over- or under-represented at certain points depending on the sample. This is exemplary shown in Figure 4.1, where the selected sample consists only of some of the smallest and largest

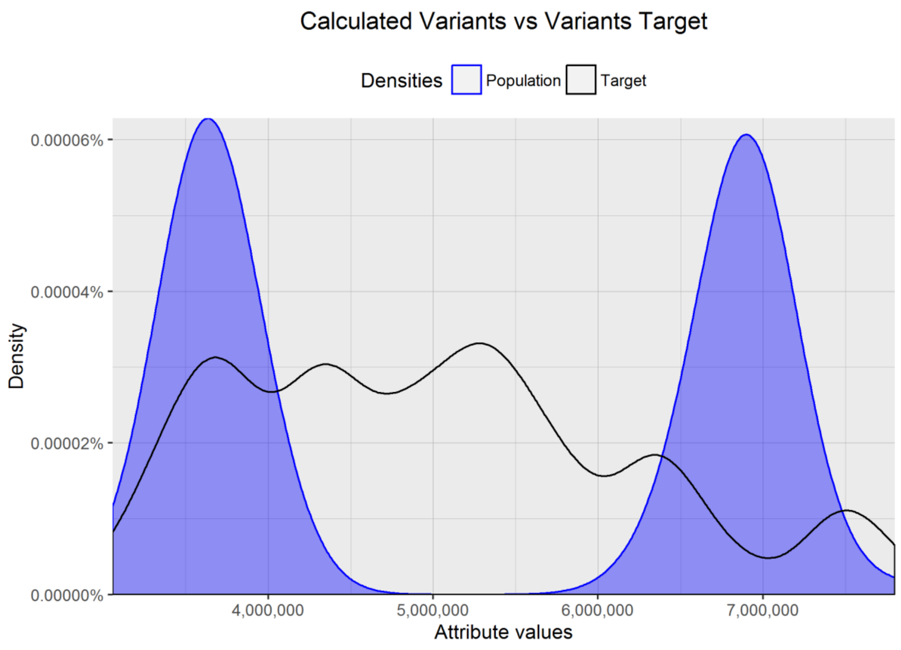


Figure 4.1: Example of a bad variant sampling

available configuration sizes. The central part where usually the most configurations reside is left out, so that the resulting variant distribution is over-represented in the lower and upper value range, and under-represented in the center. It was created using a combination of feature-wise sampling and negative feature-wise sampling, which tend to oversample small and large configuration sizes.

The calculation of correct samples, who conform to the variability models specifics is therefore essential to the success of the approach. The more variants are generated the more time and resources it takes to compute the variant distributions. This creates a tradeoff between the accuracy with high amount of variants versus the available time and resources. To choose an optimal number of variants it is necessary to create sampling techniques, which allow the precise setting of the amount of maximum desired variants.

4.3 Problem Encoding

As shown in Section 2.5.1, the encoding of the chromosomes for the genetic algorithm is essential to the success of the approach. It is a mapping from the search space of all potential solutions into a form that a machine can process. The selection of a proper encoding is the starting point for solving problems with genetic algorithms. It determines which genetic operators are applicable to a problem and defines the foundation for the objective function and the fitness calculation.

For the annotation of variability models with realistic attribute values, an encoding is needed which transfers the assignment of feature and interaction attributes to features and feature-combinations into a format which is computable by the genetic algorithm. Possible alternatives are, the binary encoding, where 1 (or True) represents the presence of a feature in a configuration. This brings the question forward on how to include interactions: are they part of the encoding or do they need to be captured in a post computing step at the fitness calculation? The second alternative is the value encoding, where the value of a chromosome specifies a feature's attribute value. Interactions can easily be added at the end of the encoding, as we know the exact amount of features and interactions beforehand. These alternatives need to be prototyped against, in order to find a proper design of the genetic algorithm.

4.4 Fitness Function

The fitness function assigns a quality measure to a solution candidate. It is the central part of the genetic algorithm as it determines which candidates are further selected for reproduction. The fitness evaluation guides the genetic search to the targeted result, it defines the problem to solve in the evolutionary context. Therefore, it must be able to accurately define a solutions quality. Because the fitness function is the main part in the iterative process of the genetic algorithm it should be computationally cheap and scalable to large problems sizes.
For the generation of large attributed variability models, the quality of a solution candidate is defined as the similarity of the feature, interaction and variant distributions to the targeted real-world attribute distributions. The fitness function must be able to compute a similarity measure of two value distributions of the same size. Different methods exist for the calculation of such a similarity measure, ranging from statistical tests such as the two-sample Kolmogorov-Smirnov test or the Cramér-von-Mises test, to different distance metrics as the Chi-Squared distance or the Euclidean distance. Each technique has different properties in precision and computational effort which have to be evaluated against each other.

5. Approach

This chapter describes all necessary steps for the creation of attributed variability models with realistic attribute values and interwoven interactions. After a general overview we provide a detailed description of each step and the prevalent problems we tried to overcome with our solution in the subsections.

An graphical overview for the generation of an attributed variability model with our tool is presented in Figure 5.1.

Input At first, the user has to provide a non-attributed variability model. The tool has parsers for SPLConqueror models, BeTTy attributed variability models (.afm), Dimacs models as used in the LVAT, and the simple exchange format used in the SPLOT (.sxfm). Additional parsers can easily be added to the implementation. Next, the user has to define the amount and order of interactions. Most importantly the user has to select the desired target distributions for the features, interactions and system variants. One can choose between the distributions from the SPLConqueror dataset, a normal or uniform distribution, or specify own comma separated values as input. At last, the user has to

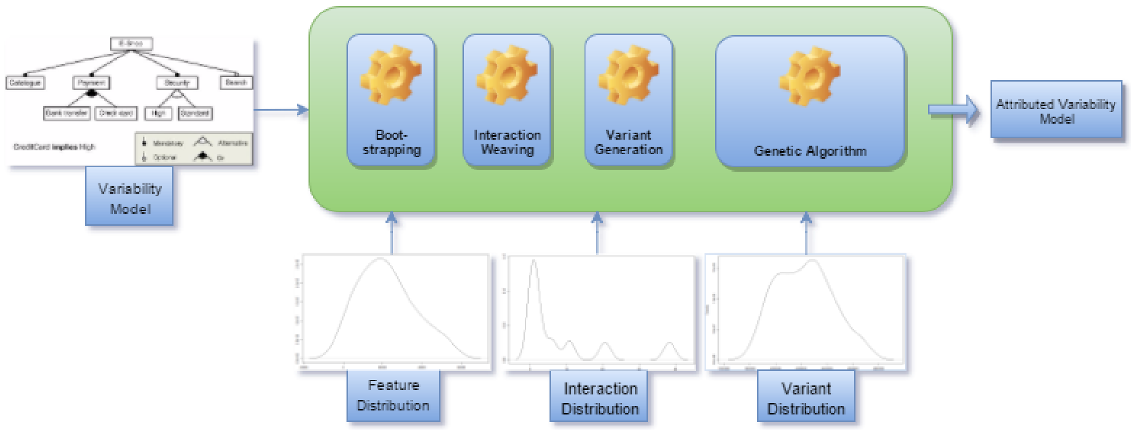


Figure 5.1: Overview of the approach and the involved parts

define the variant sampling strategy and the parameters for the genetic algorithm, before the required settings are complete and the optimization process can be started.

Bootstrapping The next step is to bring the selected input distributions into the correct size and range. We bootstrap the input feature distribution to the size of the feature count of the variability model. The input interaction distribution is bootstrapped to the amount of desired interactions. In case the input interaction distribution is selected from a different system or non-functional property as the feature distribution, we additionally rescale the value range of the interaction distribution to the value range of the to the feature distribution corresponding interaction value range. For example, if we select the BerkeleyDB performance feature distribution and the LLVM performance interaction distribution, we then rescale the bootstrapped LLVM interactions to the value range of the BerkeleyDB performance interaction values to even out eventual inequalities.

Interaction Weaving If the user has chosen to add interactions to the variability model, we choose pairs, or combinations of more than two features depending on the interaction order, at random until the amount of desired interactions are met. No combination of features can be selected twice so that each interaction is unique. We additionally check the interaction for satisfiability with the variability model, such that only valid combinations of features are generated as interactions. The interactions are stored in a compact matrix which describes the partaking features for each interaction.

Variant Sampling In this step the user can choose from different sampling strategies for the generation of variants. These options range from the heuristics implemented in the SPLConqueror tool like the Pairwise, Feature-wise or Negative Feature-wise heuristic to pure random selection and configuration size sampling techniques, which will be explained in the subsections in detail. The generated variants are saved in a Boolean matrix which describes which features are present in the configurations.

Genetic Algorithm The problem is encoded as a real-valued array with the size of the features in the variability model plus the amount of interactions in the end. It declares the attributes value of the feature distribution and the interaction distribution as the search space for the chromosomes in the genetic search. A minimal and maximal value can be assigned individually to features and interactions. The system variant attribute values are calculated from these two distributions and the feature-variant- and feature-in-interaction matrix.
We chose the NSGA-II multi-objective optimization genetic algorithm for our implementation. This allows us to use the three optimization objectives (feature,interaction and variant similarity) instead of combining the fitness calculation into one single objective.

Fitness Calculation The fitness of each objective is calculated by one of the four implemented distance metrics/test-statistics described in the subsections. It provides a metric of the goodness-of-fit of an individual solution candidates feature and interaction distribution to the selected targeted distributions. It computes the matrix multiplication of the feature-variant matrix and the solution candidates feature-distribution. Then a matrix-multiplication calculates the total interaction values of each configuration. Lastly the interaction values for each configuration are added to the result of the feature-variant multiplication result. This provides the total variant distribution of the solution candidate, which is then in turn compared to the targeted variant distribution. Each similarity measure is then applied as the fitness values for the three objectives.

Selection Operator We use a binary tournament selection with the tournament size 2.

Crossover Operator We chose the simulated binary crossover operator (SBX), which is a special case of one-point crossover for real-coded variables.

Mutation Operator Polynomial mutation simulates the bit-flip mutation on real-valued decision variables.

Solution Selection The genetic algorithms returns a set of pareto-optimal solutions. To return a single solution we use a weighted sum selection.

The following sections explain the steps of the process in greater detail.

5.1 Variability Model Parsing

At startup, the tool expects a non-attributed variability model as input and parses it into the SPLConqueror format. Each feature is represented as a configuration option which can either be selected or deselected and holds a number of constraints. Those include parent-child relations, exclude and imply options. Additional cross-tree constraints are saved in the variability model. We implemented parser for the BeTTy feature model format, SPLOT's simple exchange format (SXFM), Dimacs models, as well as the already existing SPLConqueror model parser.

5.2 Inputs

After parsing the variability model, the tool requests the amount and order of interactions. The amount of interactions can either be provided by a fixed number, or by a percentage of the amount of features in the variability model. The order of interactions must be given by percentages, for example 70% first order, 20% second order, and 10% third order interaction. The sum of order percentages must sum up to 100. It also possible to specify zero interactions, which will exclude them from the process and creates an attributed variability model without interactions.
The next step is to select the targeted feature distribution. It is possible to choose from one of the included feature distributions from the SPLConqueror dataset, or to create a new one according to a random function. We implemented the normal distribution and the uniform distributions. The normal distribution must be given with mean and standard deviation. The uniform distribution is taken from a minimum and maximum value.

After target distribution was selected, it needs to be rescaled to the variability models amount of features. As stated in Section 4.1, we need to estimate the underlying probability distribution of the input sample using kernel density estimation. This is done by an `C#` to R interface, where we use the Kernel Smoothing[1] package for kernel-density estimation. It enables the calculation of the estimate with the optimal bandwidth of the underlying kernel using a plug-in bandwidth selector.
Next a random sample is drawn with the size of the variability models feature count. To evaluate the correctness of the random sample, it is necessary to conduct a test for the goodness-of-fit of the drawn random sample, and the targeted feature distribution. This is done using the two sample Kolmogorov-Smirnov test of the Accord.NET framework.[2] The function returns a test-statistic which defines the similarity of the two distributions. The density-estimation, random sample drawing, and the test can be performed multiple times, to receive the best possible random sample. Additionally, we plot the drawn random sample, and the selected target feature distribution, which allows the visual inspection of the bootstrapping quality. An example is given by Figure 5.2, the green distribution represents the drawn random sample. The black line represents the targeted feature distribution.

The same process is used to obtain the interaction value distribution. The only difference is that the selected interaction distribution is bootstrapped to the amount of desired interactions. Lastly the user needs to specify the targeted variant distribution, which will then be bootstrapped to the amount of generated configurations.

5.3 Interaction Weaving

For the creation of interactions, we require the amount and order of desired interactions. We randomly select two or more (depending on the order) features and check if the selected features are a valid subset of the variability model with the help of Microsoft Solver Foundation wrapper found in SPLConqueror. In case the selected features are not compliant to the model, for example because of an exclude constraint between two selected features, we repeat the step and draw a new random interaction. If it is a valid configuration we add a randomly selected interaction value to it. This process is then repeated until the amount of interactions are reached. We save the interacting features such that no interaction occurs twice. However higher order interactions can consist of an already included interaction of lower order.
The interactions are then added to the SPLConqueror variability model, which maintains an influence model of attribute and interactions. To prepare the interactions for the genetic algorithm we transform the generated interactions into a compact Boolean matrix representation. For each interaction (rows) we mark the partaking features (columns) with true.

5.4 Variant Generation

As shown in Section 4.2, the sampling of an appropriate subset of variants is needed for the calculation of the distribution profile over all system variants. We reuse

[1] https://cran.r-project.org/web/packages/ks/ks.pdf
[2]http://accord-framework.net/

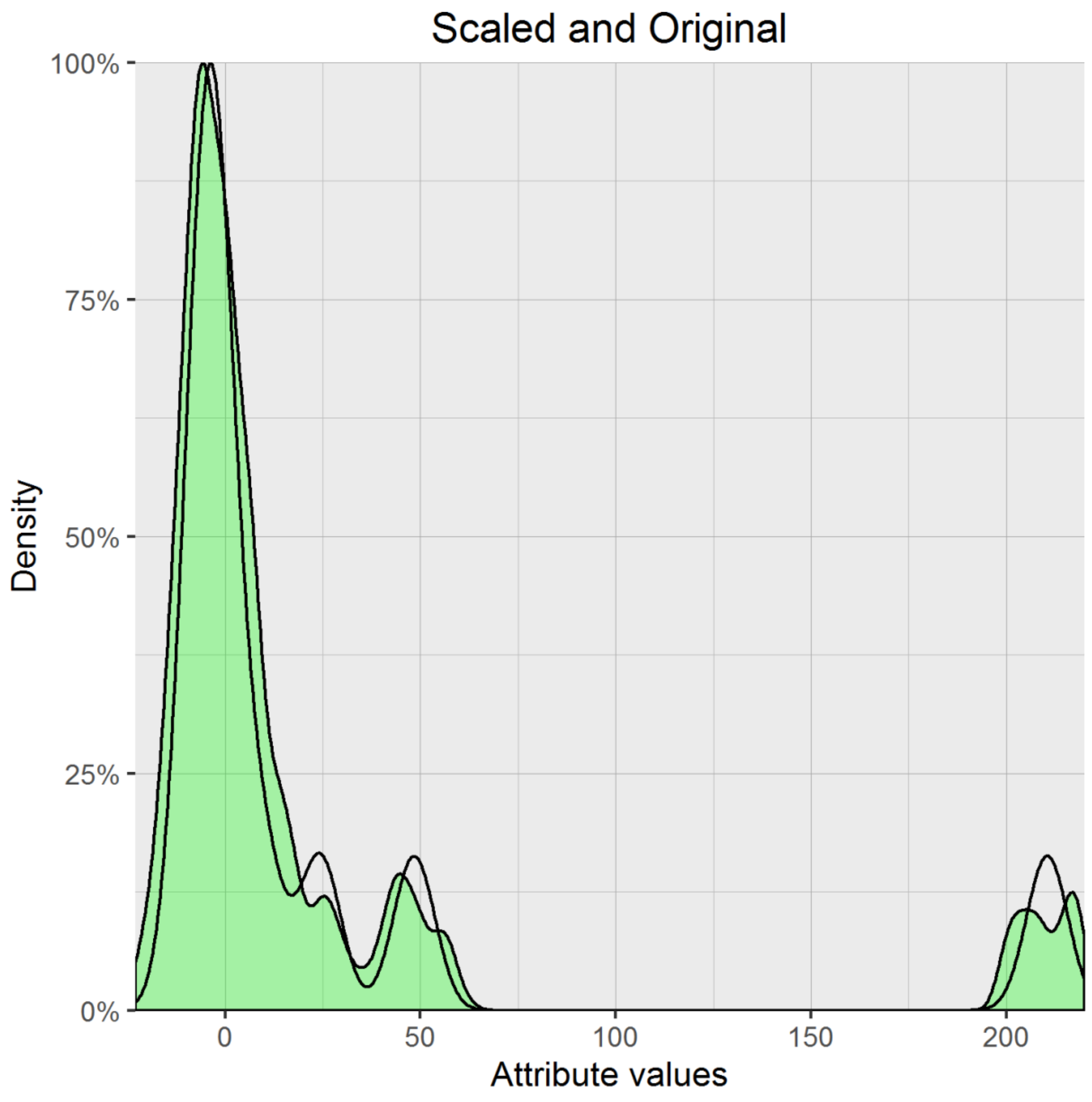


Figure 5.2: Example of a drawn random sample after kernel-density estimation in comparison with the targeted feature distribution

the existing SPLConqueror interface to the Microsoft Solver Foundation for the creation of valid variants from the variability model. Next to the existing heuristics from SPLConqueror such as the Pairwise, Featurewise, or Negative-Featurewise, we created heuristics which apply random generation on the feature-count level in different ways. The reasoning behind this is that without constraints the variability model maximum amount of possible variants follows a binomial distribution. There exists one minimum and one maximal configuration with all possible features, in between the possible amount of configurations increases until half of the amount of features and decreases afterwards. The constraints then alter the distribution from the binomial distribution in such a way that the real, valid amount of configurations for each feature count can not be easily figured out. Working out the exact distribution of valid configurations of arbitrary variability models is not part of this thesis. The following sections present the implemented heuristics for configuration size sampling.

5.4.1 Configuration Size Random Sampling

The following heuristics try to create random configurations from the complete spectrum of the variability model. In order to achieve this, we add an additional constraint to the CSP-Solver. This addition directs the solver to only search for configurations which consists of a certain count of features. This is implemented using the Solver Foundations M out of N operator. We can then generate valid configurations which have the desired count of features. The amount of random samples with the selected feature count varies in the implemented heuristics and is adjustable. This allows the precise definition of the maximum number of generated variants, which is important in order to create as many variants as possible without exceeding the machines available memory and processing power.

Fixed Count

In the fixed count heuristic, the user is able to specify a constant count of configurations for each configuration size of the variability model. In figure Figure 5.3 a constant value of 50 is selected. For each configuration size from one to the size of the variability model, we generate 50 configurations at most. The maximum amount of configuration is given by the selected constant value times the feature count of the variability model.

Linear

In the linear heuristic, the user must also define a constant count of configuration, but the number of configurations increases linearly up to the first half, then decreases linearly until the greatest configuration size is reached. The maximum amount of configurations, where c is the selected constant and s is the variability model size, is given by:

$$Max\,Configs_{linear} = \sum_{x=1}^{s} \begin{cases} x + c, & i \leq \frac{s}{2} \\ s - x + c, & i > \frac{s}{2} \end{cases}$$

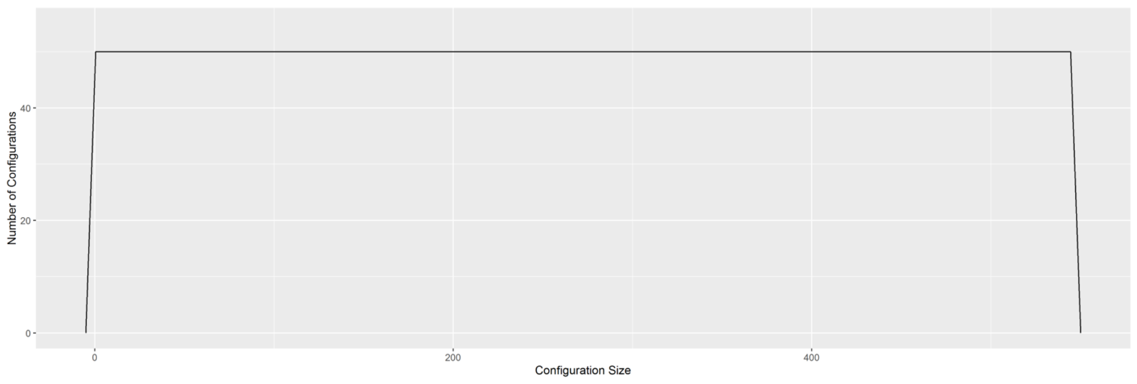


Figure 5.3: Variant sampling visualization with fixed configuration size

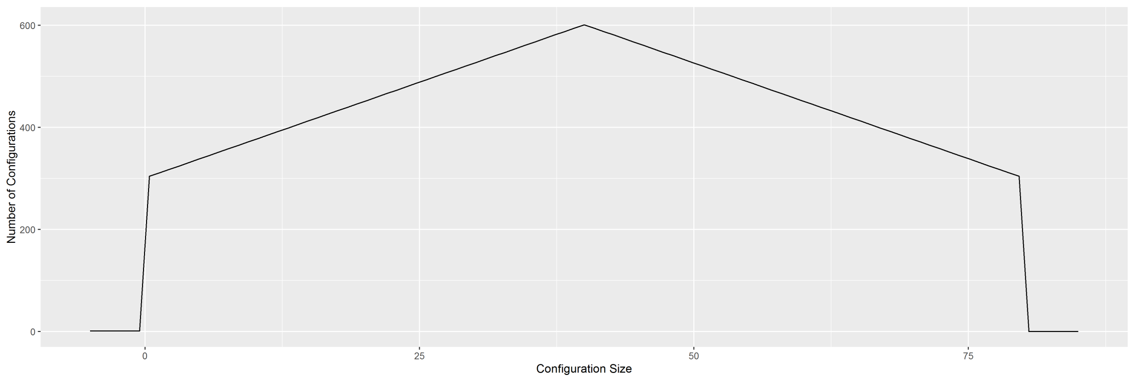


Figure 5.4: Variant sampling visualization with linear configuration size

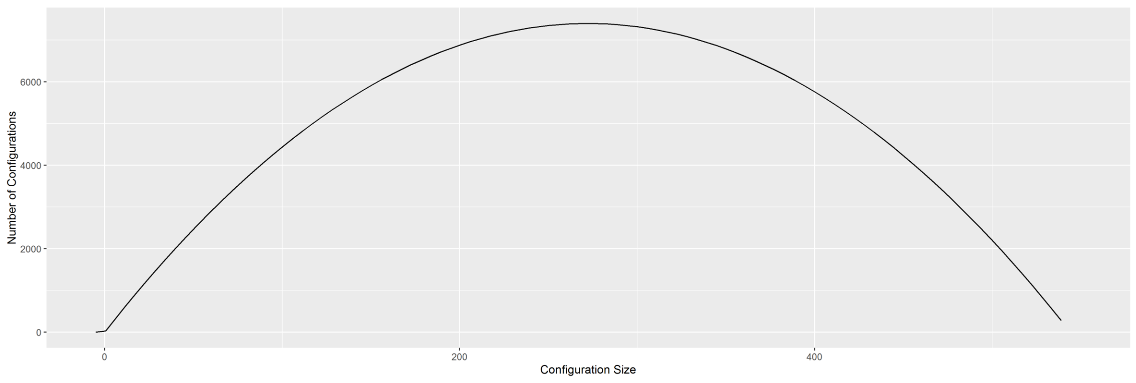


Figure 5.5: Variant sampling visualization with quadratic configuration size

Quadratic

In the quadratic heuristic the user must define a value between zero and 1, which controls the curvature of the quadratic function. The function is shifted to the right so that the maximum is at the half of the amount of features. The roots of the quadratic function are at zero and the maximum amount of features. The amount of configurations increases quadratically until the half of the maximum configuration size is reached, afterwards it decreases quadratically. The maximum amount of configurations, where *a>0* is the curvature and s the variability model size, is given by:

$$Max\,Configs_{\,quadratic} = \sum_{x=1}^{s} -a(x-\frac{s}{2})^2 + \frac{as^2}{2}$$

5.5 Calculation of the Feature/Variant and Interaction Matrix

From the previous steps we obtained the generated configurations and the created random interactions. For the fast processing of these information in the genetic algorithms fitness evaluation we transform it into two compact matrix representations. The feature matrix describes which features are present in each configuration. The rows are built from the generated configurations, the columns from the features of the variability model. We set the value of mth row, and nth column to one, if the nth feature is present in the mth configuration. This process is done in parallel, as the amount of configurations is desired to be large.
The second matrix, the interaction matrix, stores the information of the created interactions per configuration. We set the mth row, and nth column to one, if the nth interaction is present in the mth configuration. This is also done in parallel. A brief example of these matrices can be seen in Figure 5.8. The usage of the matrices is explained in the next chapter.

5.6 Genetic Algorithm

This section describes the main part of the tool, the genetic algorithm. Its main goal is to find an assignment of feature and interaction attribute, which is similar to the specified target distributions, such that the resulting variant distribution also resembles the specified target distribution of variants. The inputs for the genetic algorithm are:

- The desired feature, interaction, and variant target distributions
- The feature and interaction matrix from the generated configurations and interactions.
- Genetic algorithm settings like the population size, maximum amount of evaluations, probability of mutation and crossover operators and distribution indexes.
- The selected test statistic and their special settings.

We selected the NSGA-II multi-objective genetic algorithm created by Deb et al.[DPAM02]. It is an elitist, non-dominated sorting genetic algorithm. Elitist refers to the method of transferring a certain amount of good individuals, the elites, into the next generation of the population. Non-dominated sorts the solution candidates into levels of pareto-optimal fronts. The first front consists of the pareto optimal solutions. The second front consists of the pareto optimal solutions minus the ones from the first front and so on.
Our tool is using the implementation from the C# port of the popular jMetal[DN11] Framework called jMetal.Net.[3] A visualization of the main loop for the parallel version is presented in Figure 5.6.

At the initialization new random individuals are created until the population size is reached and their fitness value is evaluated. The fitness calculation is explained in Section 5.6.1. The main loop consists of two parts, at first the selection and creation of new candidates through the genetic operators and their fitness evaluation. A second pool of offspring population is maintained in each iteration. It is filled from children created from the crossover-operator and mutation from two tournament selected parents. Afterwards the offspring populations fitness is evaluated. In the second phase, the union of the population and the new offspring population is created, and ranked according to non-dominated sorting, which splits the population into pareto fronts of dominating solutions, where the first front is the pareto optimal front. The population is then filled with the individuals from the best fronts. In case a front is too big, for example when the front has more individuals than the remaining spots in the population, then only the best individuals from this front are chosen by their crowding distance. The crowding distance prefers individuals which are farther away from other individuals in the solution space to promote spread in the population and to avoid clustering of similar individuals. Figure 5.7 illustrates the selection process for the next generations population.

This procedure is repeated until the maximum amount of evaluations is reached. Lastly the final populations best front is returned.
The implemented problem encoding is a real value encoding. The mapping for an attribute assignment to features and interactions is implemented as an array of double values with the size of the amount of features plus the amount of interactions.

[3]http://jmetalnet.sourceforge.net/

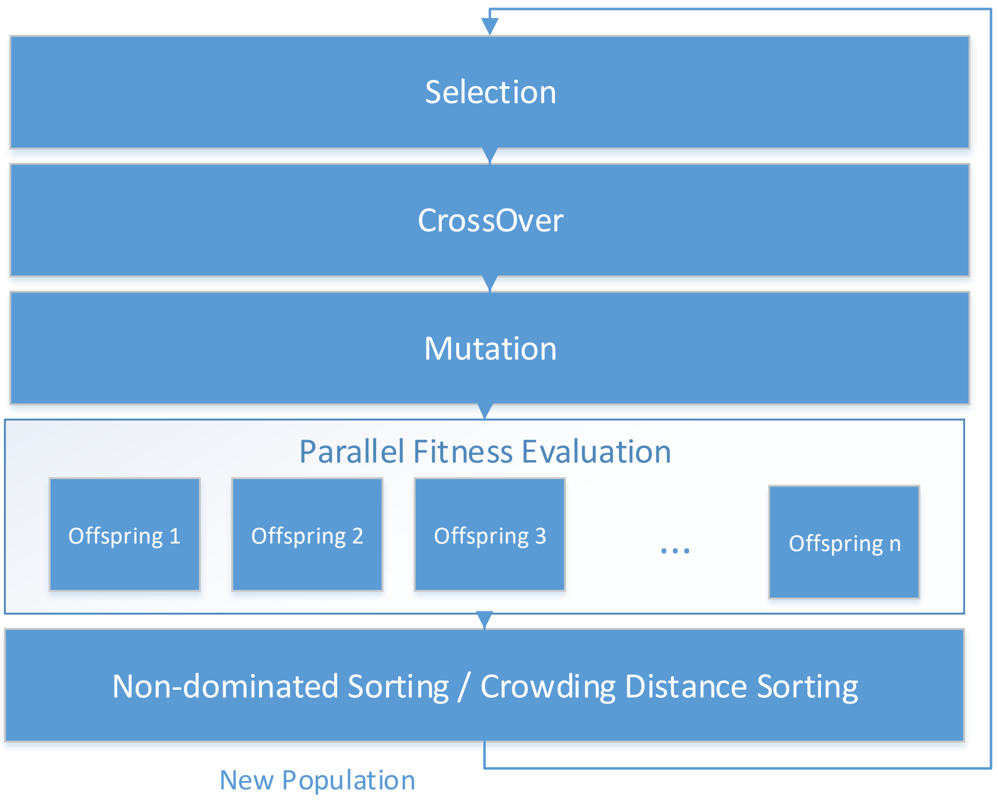


Figure 5.6: Main Loop of the Parallel NSGA-II algorithm

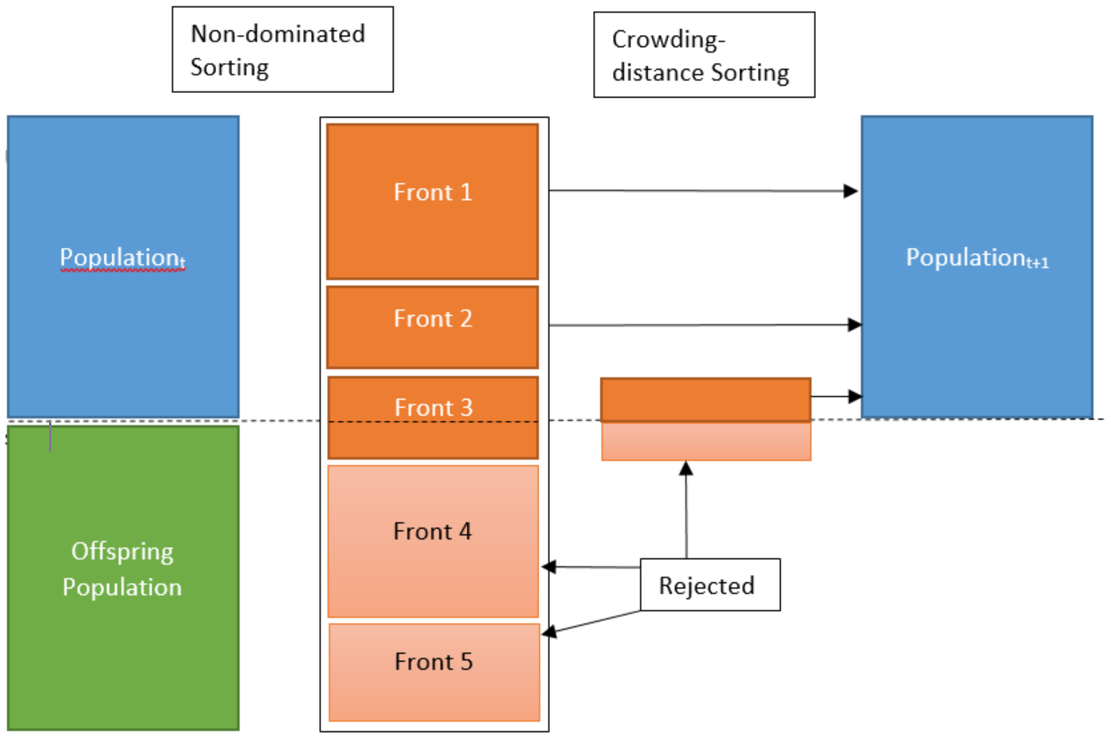
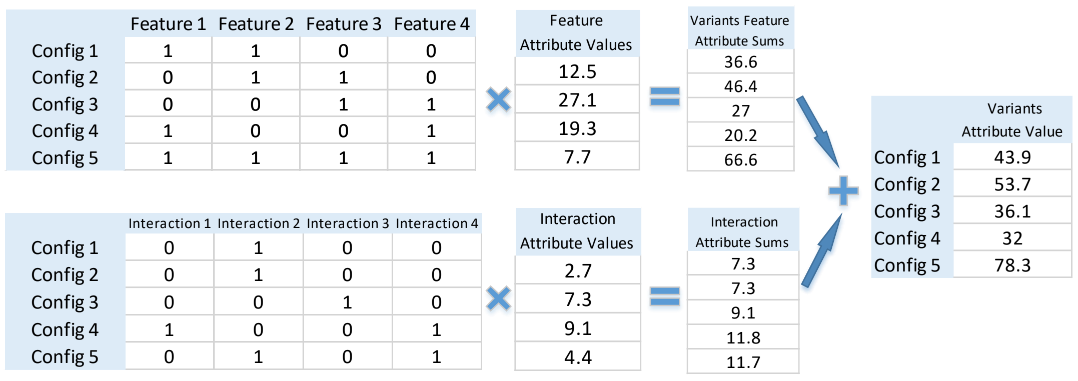


Figure 5.7: NSGA non-dominated and crowding distance sorting



	Feature 1	Feature 2	Feature 3	Feature 4
Config 1	1	1	0	0
Config 2	0	1	1	0
Config 3	0	0	1	1
Config 4	1	0	0	1
Config 5	1	1	1	1

×

Feature Attribute Values
12.5
27.1
19.3
7.7

=

Variants Feature Attribute Sums
36.6
46.4
27
20.2
66.6

	Interaction 1	Interaction 2	Interaction 3	Interaction 4
Config 1	0	1	0	0
Config 2	0	1	0	0
Config 3	0	0	1	0
Config 4	1	0	0	1
Config 5	0	1	0	1

×

Interaction Attribute Values
2.7
7.3
9.1
4.4

=

Interaction Attribute Sums
7.3
7.3
9.1
11.8
11.7

+

	Variants Attribute Value
Config 1	43.9
Config 2	53.7
Config 3	36.1
Config 4	32
Config 5	78.3

Figure 5.8: Sample calculation of variants attribute values

The first value represents the assignment of an attribute value to the first feature, the second value to the second feature and so on. Afterwards follow the assignments for interaction attribute values in the array. A possible solution candidate therefore consists of the attribute values for the features and interactions.

5.6.1 Fitness

In our tool a solution candidate's fitness is calculated by their similarity of the feature, interaction and variant distributions to the targeted distributions. A solution candidate consists of attribute assignments to the features and interactions. The resulting variant distribution is calculated with the help of the feature and interaction matrices. This process is illustrated in Figure 5.8. We perform two matrix multiplications. The first calculates the sum of the features for each configuration, by multiplication of the solution candidate's feature part and the feature matrix. The second matrix multiplication calculates the influence of the interactions for each configuration, by multiplication of the interaction matrix, with the interaction attribute part of the solution candidate. This results in two values for each configuration, which are added to receive the total attribute value for the configuration. The similarity of the distribution pairs is then calculated with one of the test-statistics/distance metrics explained in Section 2.6.

For the calculation of the variants fitness some preconditions have to be respected depending on the selected test-statistic. First of all, the value range of the targeted variant distribution has to be adjusted in order to be comparable with the calculated variant distribution. This is necessary since the varying amount of features and value ranges for the feature assignments results in different value ranges of variant distributions. This is done by the linear transformation function, where we transform the values x from the range [A,B] to the range [C,D]:

$$f(x) = C(1 - \frac{x-A}{B-A}) + D(\frac{x-A}{B-A}) \tag{5.1}$$

Secondly, some test-statistics such as the Cramér-von Mises Test require, in addition to the same value ranges, the same sample size. This is solved with the bootstrapping of the targeted variant distribution to the amount of generated variants.

The calculated distance of the distribution pairs is then directly inserted as the solutions fitness. The NSGA-II's optimization goal is the minimization of the fitness value, which adheres directly to the minimization of the distance between the current individual and the target distributions.

5.7 Solution Evaluation

The output of the NSGA-II algorithm is the first front of solution candidates, which is the pareto-optimal front. As this front's size can be equal to the selected population size, we need a technique to select one solution from the front.
We chose a weighted sum selection technique because of the easy understandability and the simplicity of the implementation. A weight w_i is set for the ith objective, where $\sum w_i = 1$. The fitness values are normalized to the range [0, 1].
The selected solution is the candidate with the minimal weighted cost, where w is the weight, f the fitness, n the amount of objectives:

$$c_w(x) = \sum_{i=0}^{n} w_i \cdot f_i \tag{5.2}$$

The weighted sum selection process is visualized in Figure 5.9. It shows a fabricated 2-dimensional pareto-front of normalized feature and variant fitness values. With 2 objective values, the weighted sum selection is visually equal to the construction of a straight line, with the negative quotient of the selected weights as slope. The Y-intercept is then minimized until we found the minimal solution. The green and orange line describe two different weight assignments, one with equal weights (50:50) and one with double weight on the variants fitness. This leads to a selection of a solution with lower (better) variant fitness, but higher (worse) feature fitness.
This simple procedure, which works the same for higher dimensions, allows us to specify our desired solution with a high amount of precision.

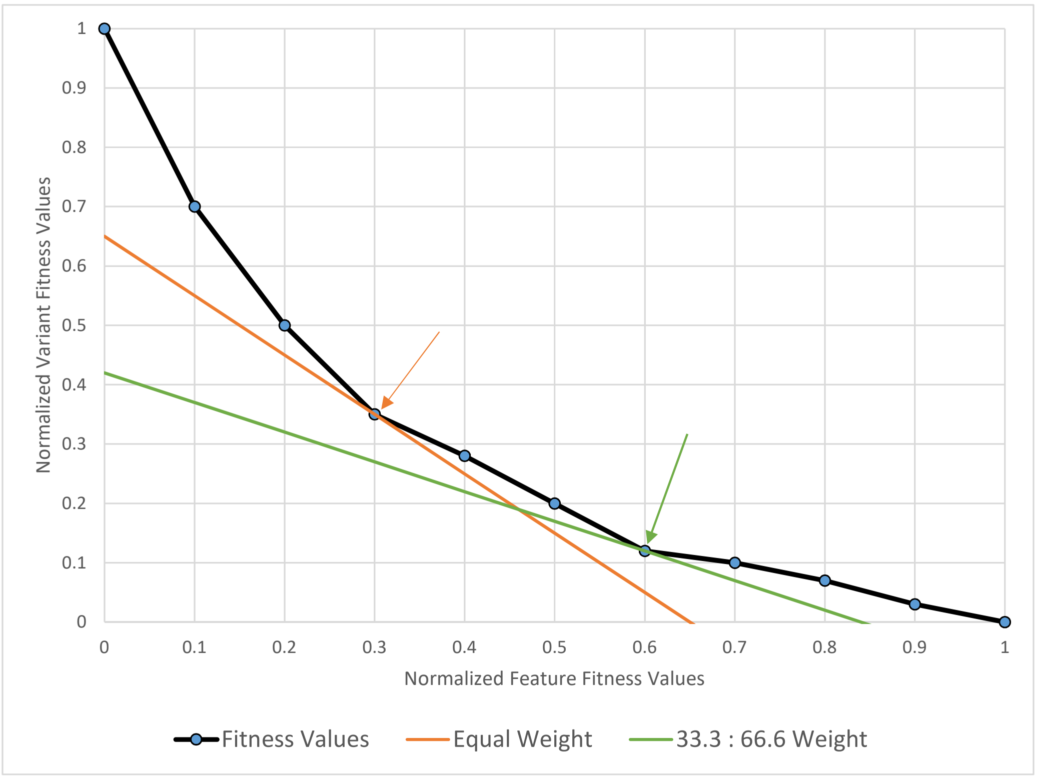


Figure 5.9: Solution selection from the pareto-front visualized with a two-objective example problem. Two different weights and the resulting selection of solutions are shown

6. Evaluation

The tool implementation allows many different settings for the creation of attributed variability models. Starting from many selectable input distributions with their special characteristics, over different variant sampling techniques, up to selectable fitness calculations and settings for the genetic algorithm. In this section we want to investigate if the variety of different options is really necessary to receive good solutions, or if single settings always outperform other options and thus making them redundant and unnecessary. In particular we want to answer the posed research questions from the introduction and split them into detailed operationalized research questions to provide informative and sound answers to the questions.
After an overview over the tool's general scalability, we investigate the effects of an implemented early stop criterion.
In order to answer **TI RQ1**, the existence of a single best fitness metric for all distributions and problem sizes, we perform multiple experiments on the performance and solution quality of the four implemented fitness metrics. The results and discussion can be found in Section 6.2.
TI RQ2 focuses on the effect of different variant sampling strategies on the results of the solutions. It is clear that the more variants there are the more accurate the system variant distribution will be, but restrictions in time and available computing resources pose a limit to the amount of derivable system variants. The goal is to find an optimal sampling strategy which can handle the trade-off between accuracy and the physical limits of the machine as best as possible. The experiments and results for this question are explained in section Section 6.3. All experiments were performed on a Intel i7-4790 4-core (8 threads) 3.60GHz, 16Gb RAM, Windows10 Desktop PC. If not explicitly mentioned, the parallel version of the NSGA-II algorithm was chosen.

6.1 General Tool Evaluation

A great focus with the tool was set on the scalability and parallelization of tasks to allow the handling of large attributed variability models.

6.1.1 Tool Performance Evaluation

For the overall performance we focused on evaluating which components of the tool uses up most of the program's runtime, and how well they scale with increasing problem sizes.
We identified the four most time-consuming operations and measured the time spent at this step in the process. The four most time consuming operations are:

- The generation of variants from the variability model by the CSP-Solver. Has to be performed only once.
- The calculation of the feature matrix from the generated variants and the interaction matrix for the usage in the genetic algorithm. This is also done only once.
- The matrix multiplications of the feature matrix with the feature values and interaction matrix with the interaction values. Has to be performed in each evaluation step.
- The fitness evaluation with the calculated distributions from the matrix multiplications. As part of the fitness calculation it must also be done in each evaluation step.

Tool Performance: Experiment and Discussion

The most influential parameter in the processing time is the amount of generated variants. The more variants we generate the longer it takes the program to calculate a problem. Therefore, we chose to evaluate a single problem with different amounts of generated variants and measured the time spent in each component of the tool, in order to see how well the components scale with the amount of variants.
We started with 7500 generated system variants and increased the amount up to 82000 in several steps. The amount of evaluations of the genetic algorithm was set to 5000. The Cramér-von Mises test was chosen as fitness test. Each run of the tool was performed 10 times to reduce the measurement bias. The results are shown in Figure 6.1.
The measurements show that the matrix multiplication for the variants attribute distribution has the worst scaling with the number of variants. We use the Accord.Net[1] framework for the matrix multiplication. The proportion of the matrix multiplication time of the total time also increases when more evaluation steps are performed, as the variant generation and matrix creation need to be performed only once. The actual fitness evaluation and the matrix creation increases significantly slower than the matrix multiplication. The time spent in the generation of the variants can be controlled by setting a timeout for the solver after several seconds, therefore the variant generation time remains mostly constant, with only small increases.
For future optimizations the matrix multiplication can be altered from a dense matrix to a sparse matrix representation, since the feature and interaction matrix consist to a great degree of zero entries.

[1]https://github.com/accord-net

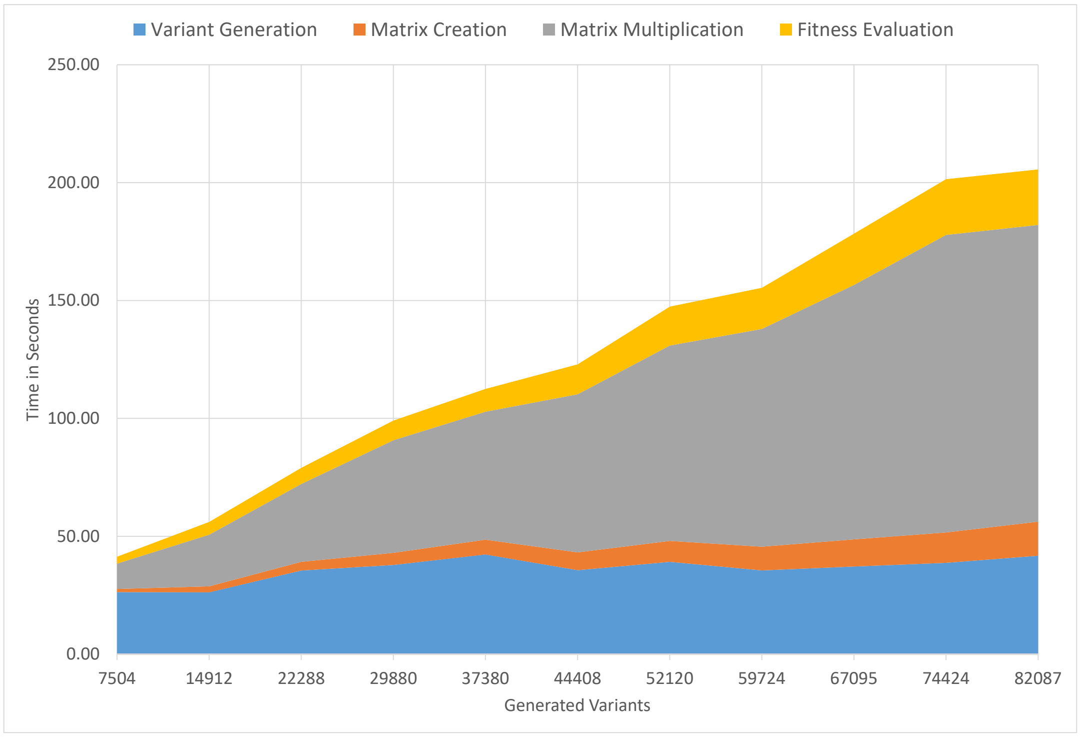


Figure 6.1: Comparison of the most time consuming operations over different problem sizes

6.1.2 Generational Distance Evaluation

The tool implements an early stop procedure which we call the generational distance criterion. The generational distance is the difference in fitness values in one evolution step between the best individual and the worst individual. If the difference in these fitness values does not pass a user-definable threshold over the course of three generations, we assume that the genetic algorithm converged to a local or global optimum and further improvements are unlikely to happen. The algorithm then stops and returns the current population. Figure 6.2 shows an example illustration of the generational distance approach. The red lines mark the difference between the best and worst individual's fitness. We performed the experiment for each of the four distribution classes, and with three different significance levels ($\alpha = [0.01, 0.05, 0, 1]$), for each test case 50 times to reduce measurement errors. Table 6.1 shows the average amount of evaluation steps needed before the generational distance did not exceed the significance levels (0.01, 0.05 and 0.1) for each fitness metric.

The results show that the euclidean metric finds a local or global optimum the fastest, while the CmV-test takes the most amount of evaluation steps. In the three-objective case the results even out, and slightly inverse with smaller significance levels.

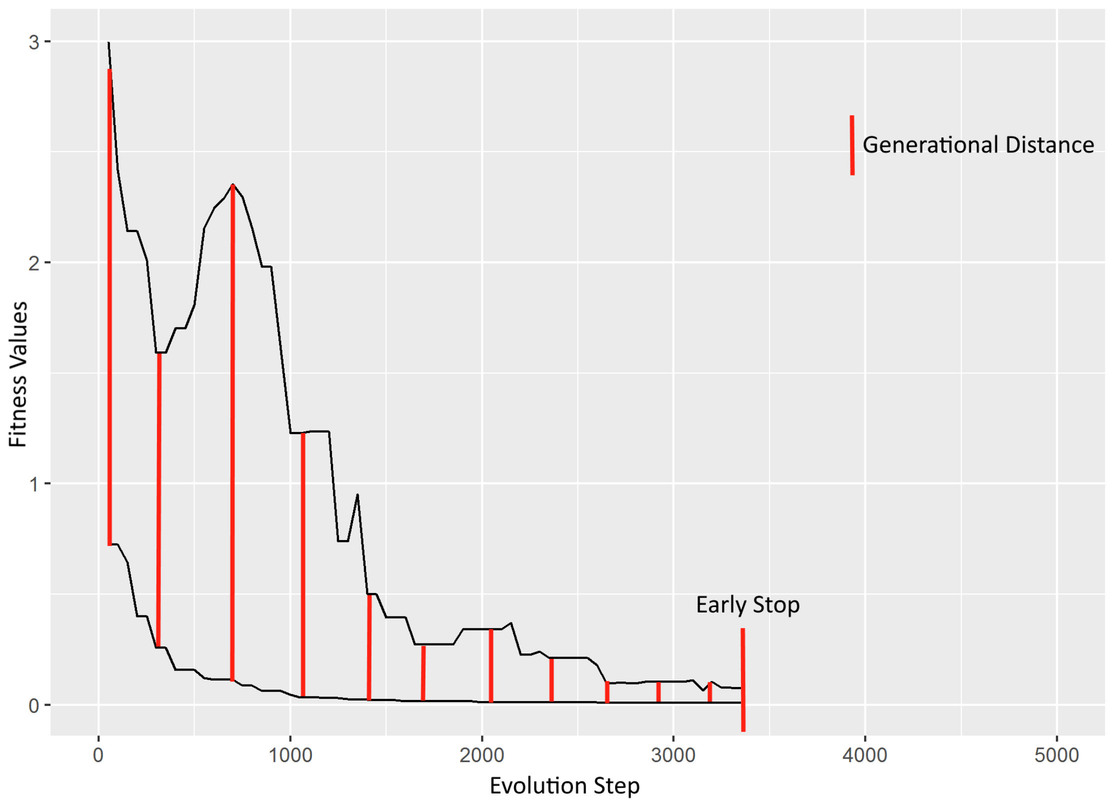


Figure 6.2: Example of the generational distance criterion. Red lines show the generational distance.

Avg. Eval. Steps at different Significance Levels				
Used Metric	Features	Significance Level		
		1%	5%	10%
Single Objective				
CMV	80	3254	3196	2569
Chi	80	1783	1840	1574
Euc	80	1478	1429	1396
CMV	290	12960	9803	2704
Chi	290	7419	5464	2505
Euc	290	4576	4071	2393
Three Objectives				
CMV	80	3694	2231	1448
Chi	80	3900	2347	1069
Euc	80	4434	2312	972
CMV	290	5981	2849	1065
Chi	290	6165	2812	1047
Euc	290	6000	2301	907

Table 6.1: Average evaluation steps of 50 runs, each with generation distance criterion active under different significance levels and fitness metrics.

6.2 Fitness Metric Evaluation

The fitness metric evaluation is split into two parts. The pure performance aspect and the evaluation of solution quality for the four implemented metrics Kolmogorov-Smirnoff test, Cramér-von Mises test, Euclidean distance, and the χ^2 distance.

6.2.1 Performance Tests

The performance part of **TI RQ1** will be answered with two newly introduced operationalized research questions:

- **Metric Performance 1**: Which fitness test is the fastest and scales the best on different input sample sizes?
- **Metric Performance 2**: Is the difference in speed significant for valid problem sizes in the tool?

Metric Performance 1: Experiment and Discussion

Experiment Setup: For each implemented fitness metric, we analyze the pure processing time of the test-statistic in a separate project. This is done because the fitness calculation in the tool is heavily parallelized so that we cannot make strong guarantees on the actual duration of a single processing task. We created standalone executables for each metric. Each executable creates two random input distributions with the size n. The test-statistic is then calculated 50 times to reduce measurement bias, and we sum up the required time for the 50 iterations, excluding the time it takes to generate the input distributions. We start with n=1,000,000 and increase it in steps of 500.000 until 10,000,000. Each time we add up the required time. Because the Euclidean and χ^2 distance are binned distances, we also include different histogram bin counts in the experiments. These include the Euclidean distance with 20 and 500 bins, and the χ^2 distance with 50000 bins. These numbers were selected arbitrarily. Additionally, we select dynamic bins where the bin count is the square root of the input sample size.
Results: The results of the experiment are plotted in Figure 6.3. It shows that the CmV test is a factor 4 to 5 times slower than the two binned distance metrics for these sample sizes. The bin count however has next to no influence on the processing time of the binned distance metrics. An explanation for the slow CmV is that it must first sort the input samples in ascending order, compute a common rank on both samples before the actual arithmetic takes places. Binned distances only need to create the histogram bins.
In Figure 6.3 the KS test is intentionally left out, because it does not scale linearly and is simple not applicable for these large input samples in fitness calculations. The KS test results can be seen in Figure 6.4, where it was used in the tool on a variability model with 80 features and increasing number of variants in comparison to the CmV test. We measured the time our tool in 10 individual runs and computed the average time.

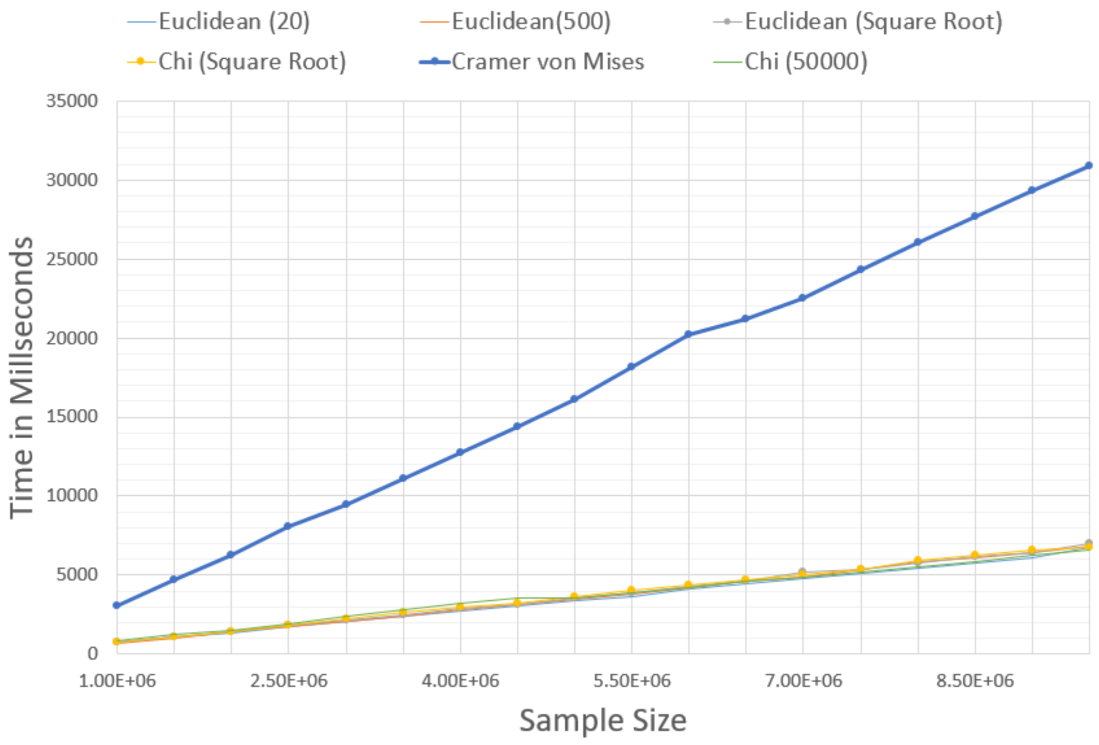


Figure 6.3: Results of the metric performance test 1. Processing time for different fitness-metrics under increasing sample sizes in an isolated environment

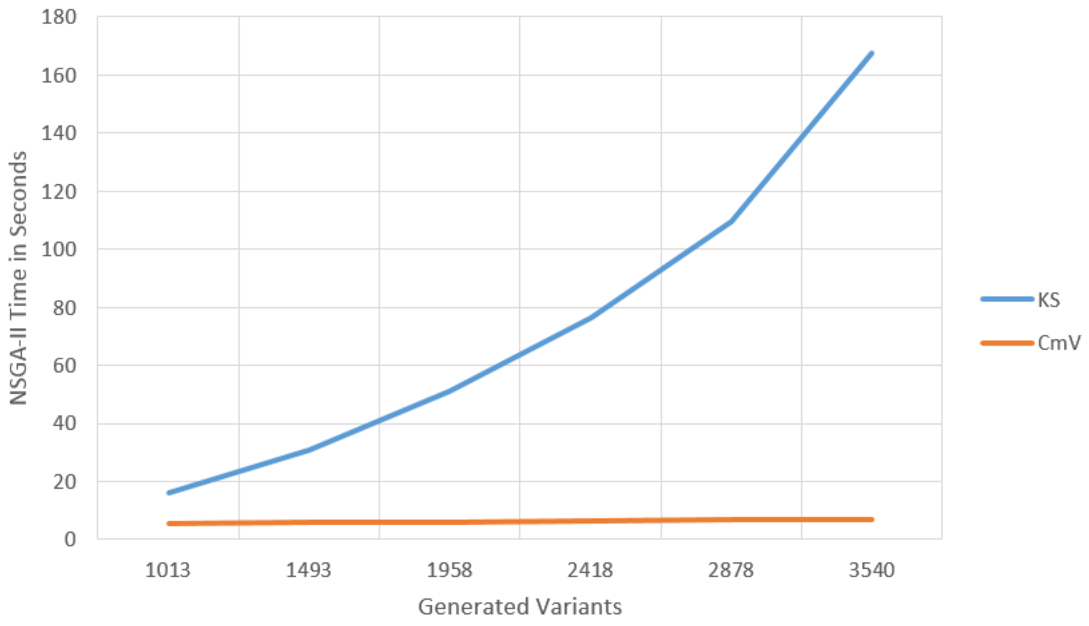


Figure 6.4: Results of the metric performance test between the KS test and the CmV test. Average duration of 10 runs with increasing number of variants

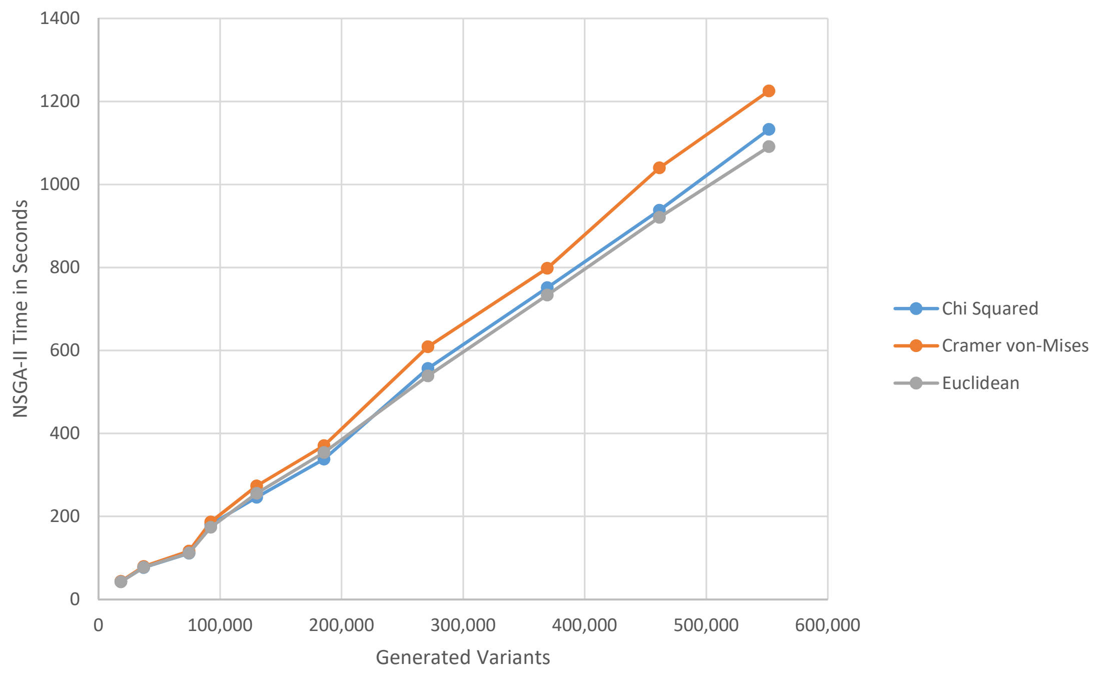


Figure 6.5: Computation time of the genetic algorithm with different test-statistics and variant sample sizes

Metric Performance 2: Experiment and Discussion

Experiment Setup: We performed multiple runs of the tool for each metric to test if the performance differences of the fitness metrics found in the first experiment also show up that clearly in actual runs of the tool.
Each tool executing time is obtained by taking the average of 10 runs. All settings remained constant except the used fitness test and the number of generated variants. We performed 5000 evaluations steps with the population size 50 of the "toybox" variability model with 545 features of the Linux Variability Analysis Tools (LVAT). We added 150 interactions and selected the BDB binary size distributions for each input distribution. The sampling method was the linear configuration size technique and the selected configuration sizes started from 50 up to 1500, which results in a generated configuration count from 18,400 to 552,000. We measured only the duration of the genetic algorithms fitness evaluation steps. The prior steps, like the interaction weaving or variant generation are not included in the measurements.
Results:
The results are plotted in Figure 6.5. The CmV test is still slower than both binned distance metrics, but in the overall duration of the genetic algorithm it results only in an increase of 10% in average up to the largest test size compared to the binned distance metrics. Of course this difference will increase with even larger amounts of variants, but it is still in an acceptable magnitude.

6.2.2 Metric Quality Tests

The second part of **TI RQ1** deals with the quality of obtained solutions from the different fitness-metrics. The main goal is to find a single metric, which delivers the

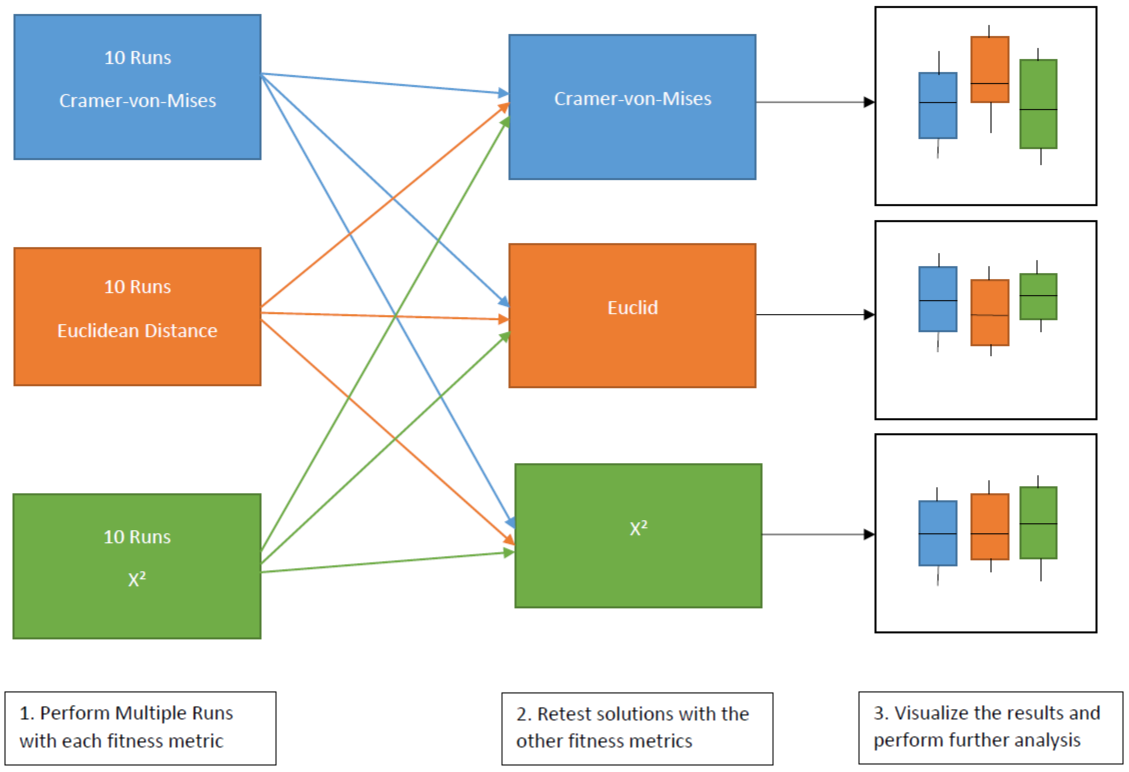

10 Runs
Cramer-von-Mises
10 Runs
Euclidean Distance
10 Runs
X²
Cramer-von-Mises
Euclid
X²
1. Perform Multiple Runs with each fitness metric
2. Retest solutions with the other fitness metrics
3. Visualize the results and perform further analysis

Figure 6.6: Cross-validation process

best possible goodness-of-fit of the resulting feature, interaction and system-variant distribution to the selected input distributions.
It is far more difficult to evaluate the quality of test statistics, because the resulting fitness values from one test-metric are not comparable to the fitness values of other metrics. Additionally, we cannot even compare the same test statistic between different sample sizes.

Metric Quality 1: Experiment and Discussion

In order to evaluate, which test statistic results in the best solutions overall, we perform multiple cross validation tests with all fitness-test metrics except the Kolmogorov-Smirnov test, which is by far too slow for larger model sizes.
The cross-validation method is illustrated in Figure 6.6. At first we perform multiple runs (10) for each of the fitness metrics under test with otherwise constant settings. With a population size of 50, we consequently receive 10x50 solutions of each fitness metric. These received solutions are then tested again with the remaining two fitness metrics. This results in 1500 fitness values of a single metric, where 500 of each descending from the 10 genetic runs of one fitness metric. This allows us to inspect how each metric evaluates the results from genetic runs performed with all metrics. The results are then illustrated as three boxplots, one for each of the three fitness metrics.
To further reason about the results, we perform a one-way ANOVA test, to determine if the influence of the selected genetic run (3 Factors: CmV, Euclidean, χ^2) is statistically significant for the calculated results (Target Variable). In case they are

significant, we further calculate the effect size (η^2). This enables us to see, if one metric performs significantly better, and if it does, we obtain a metric (effect size) on how much better it performs.
We conducted the experiment for several different distribution types and sample sizes. Starting from smaller models with 80 features up to the larger "toybox" model from the LVAT repository with 545 features. The input distributions were selected from the four general patterns we found in the dataset analysis. Since every cross-validation test results in nine different plots (3 metrics * 3 distribution types), it is not possible to show the results of every single cross-validation test, but they are included in the appendix.
A variability model with 80 features was selected from the SPLOT repository. The feature, interaction and system variant belong to the "Multi-peak" distribution pattern. The input feature distribution was taken from the performance measurements of LLVM. 40 (50%) interactions were added and the target was the BDBC performance distribution. We generated 26,215 system variants with the pseudo-random sampling heuristic, with a maximum of 500 variants per configuration size. The target variant distribution was again taken from the LLVM performance measurements. The population size was 50 and we set a maximum of 15000 evaluations steps.

Figure 6.7 shows the resulting nine boxplots. For each boxplot, the left box indicates that the genetic run was performed with the χ^2 metric, the center box shows the solution from genetic runs with the CmV test, right box is for runs done with the euclidean distance metric. The left column shows the retested χ^2 values, center column CMV, right column the euclidean ones. Top row are the feature fitness values, center row interaction values, and lastly bottom row for the variant values. Smaller values indicate a smaller distance and therefore more similar distributions to the target distribution.
We can see that the χ^2 test is valued better than the euclidean distance in almost every column. Additionally, the CmV test performed the worst in all scenarios, except when retested with the CmV itself.

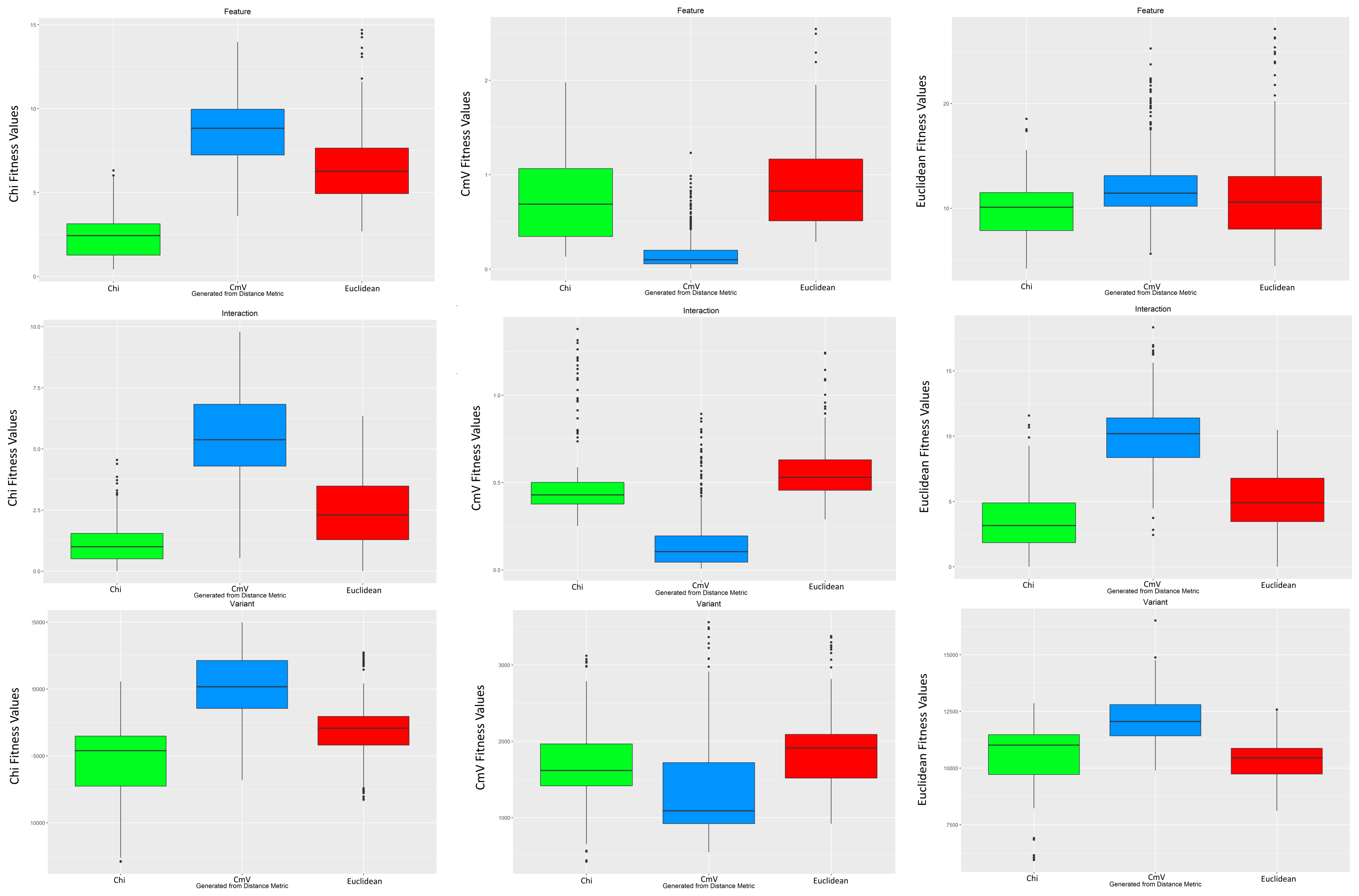


Figure 6.7: Result of the cross-validation test with the "multipeak" input distributions. Top row: Features, center row: Interactions, bottom row: Variants. Left row: χ^2 values, center row: CmV values, right row: Euclidean values. In each plot the genetic run was done with: Left box: χ^2, center box: CmV, right box: Euclidean

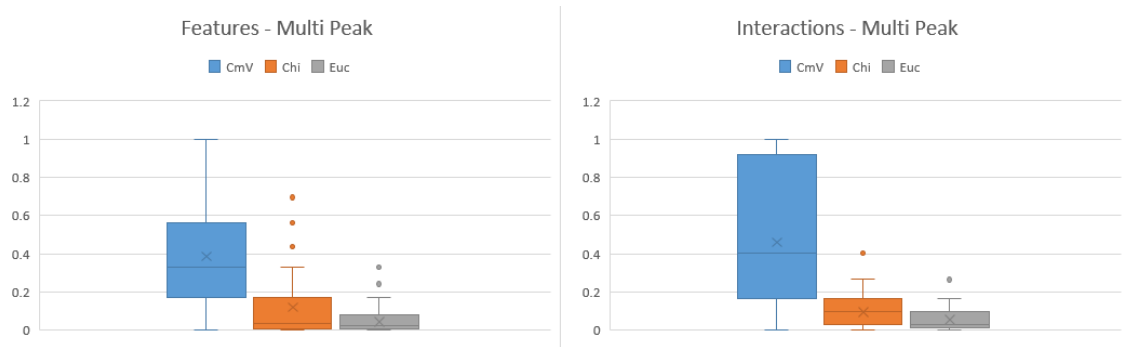


Figure 6.8: Result of the cross-validation test with the "multipeak" input distributions, using the KS-test as second evaluator. Left features, right interactions. Higher is better

From these tests we can see that the χ^2 metric performs better than the euclidean metric. But we cannot make any assumptions about the quality of the CmV test, because the results are biased in the equal combination of genetic run fitness and retesting metric. The ANOVA test was significant for each combination at a significance level of 5%. The effect sizes ranged from 0.04 (smallest effect, top right, feature distributions, euclidean values) up to 0.6 (biggest effect, top left, feature distributions, χ^2 values).
The additional tests with different distribution types and problem sizes provided similar results.
As this technique did not provide satisfactory results, we performed a second evaluation using the KS test as ground truth for the comparison of fitness metrics.

Metric Quality 2: Experiment and Discussion

The general idea of this experiment is equal to experiment 1, only that the resulting solutions are retested with the Kolmogorov-Smirnoff test. As the KS-test is a rather conservative test it was not able to provide results for the large sample sizes of the variant distributions.
Figure 6.8 shows the plots for the feature and interaction distributions. Blue boxes represent genetic runs with the CmV test, orange boxes χ^2 runs, and gray boxes visualize genetic runs with the euclidean distance metric. Higher is better, with 1 being the best value. Solutions obtained from genetic runs with the CmV test are significantly better (e.g. the target distributions and solution distribution are more similar) than the other metrics, when evaluated with the KS test. These results can be also seen for other distribution types and problem sizes.

6.3 Variant Sampling Evaluation

We implemented different variant sampling techniques. The purpose of these techniques is to reduce the amount of generated system variants, while reproducing characteristics of the variability models entire configuration distribution as close as possible. This is of course difficult because we cannot know the actual amount of configurations for each configuration size, without actually generating each system variant, which is infeasible.

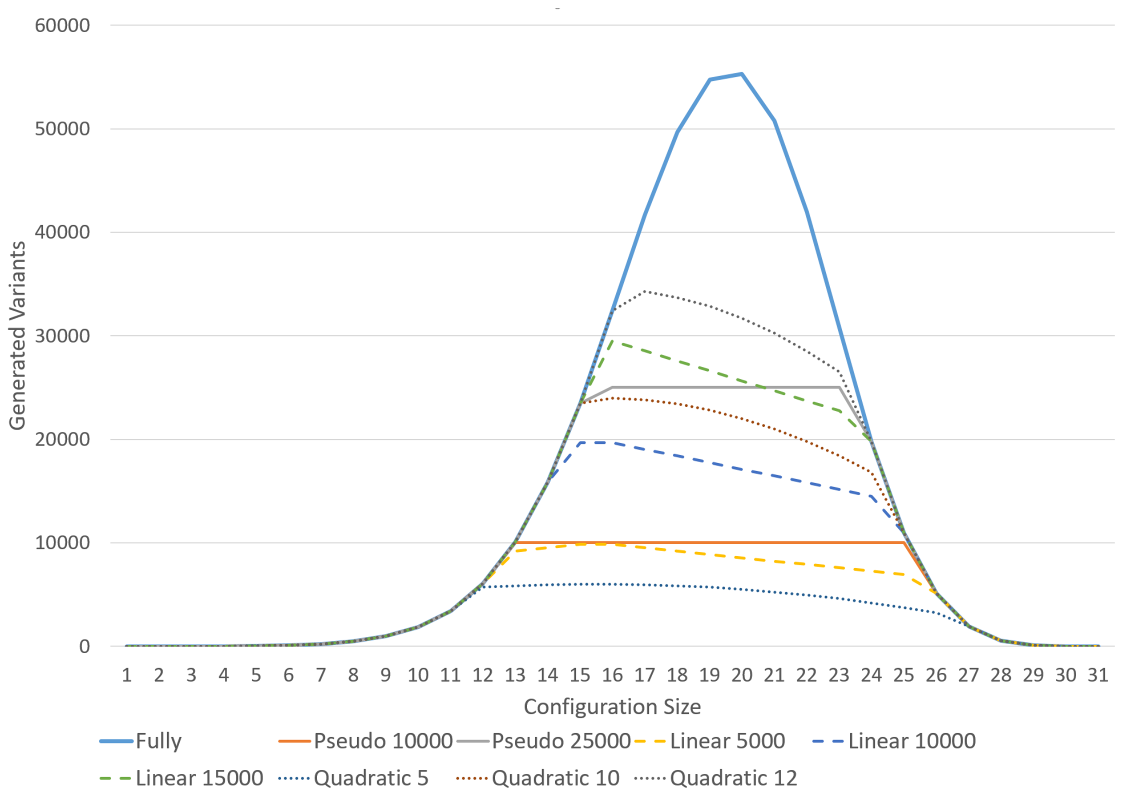


Figure 6.9: Comparison of sampling techniques of a variability model with 31 features and no constraints.

In this experiment we investigate the impact of different sampling methods on the variant result distribution, in comparison to a fully configuration where all possible system variants are generated. For this experiment we created a small variability model with 31 features. Each feature has two children (tree depth of 4) and there are no constraints at all. The fully configuration consists of 458,329 system variants. Figure 6.9 shows the sampling techniques in comparison the fully configuration. It can be seen that, the higher the setting for each sampling heuristic is set, the closer we approach the fully configuration.

Sampling Experiment

Experiment Setup We perform a run of the tool with a normal distribution (mean 10, standard deviation 2) as feature target for the same small variability model with 31 features as before. No interactions are added and no variant target is chosen, so that the resulting system variant distribution will be the pure outcome of the sampled variants multiplied with the normal distribution of the features. This enables us to visualize the effects of the sampling heuristic in a controlled environment.
We perform different runs with the same settings, only with differing variant sampling heuristics. Starting from the in SPLConqueror implemented Pairwise, Feature wise and Negative feature wise, up to the newly implemented configuration size techniques. Heuristics chosen are Fixed/Pseudo-Random 10000 and 25000, Linear 5000, 10000, 15000 and Quadratic 5 and 10, and of course the full configuration. Table 6.2

Amount of variants for different heuristics	
Heuristic	Variants
Fully	458,329
Pseudo 10000	150,876
Pseudo 25000	301,005
Linear 5000	133,299
Linear 10000	231,309
Linear 15000	310,041
Quad 5	88,244
Quad 10	273,217
Pairwise	6822
Featurewise	30
Negative featurewise	31

Table 6.2: Amount of variants generated for a variability model with 31 features under different sampling heuristics

shows the generated variants for each heuristic. The resulting distributions for the configuration size heuristics are shown in Figure 6.10 with the comparison to the full configuration. It shows how each of the heuristics is able to sample from the whole value range as the fully configuration. Each of the heuristics fails to mimic the full configurations center peak, which is acceptable because this characteristic can only be discovered when sampling fully from this exact location.
In comparison the SPLConqueror sampling heuristics Pairwise, Featurewise and Negative Featurewise should be avoided for the use in this tool, as they fail to capture the whole value range. Figure 6.11 shows the results of system variant distribution of the 31 feature variability model with the Featurewise and Negative Featurewise heuristic. This experiment was performed with a small variability model where we were able to achieve a high ratio of per sampling generated variants to a fully generation. For larger models it was not possible to generate a fully configuration with the available resources. In our multiple evaluations the Pseudo-random heuristic proved to be simple and provide reasonable results.

6.4 Evaluation Summary

The general performance evaluation pointed out that the amount of generated variants impacts the total runtime of the genetic algorithm the most, with the matrix multiplication being key factor for the runtime in each iteration of the genetic algorithm. We hope to improve the cubic scalability with the use of sparse matrices in the future.
The early stop procedure is working as intended and provides a good adaptability for the significance level and is prepared for interchangeable implementations of the stopping criterion. For future evaluations it is desirable to try out other multi-objective genetic algorithms and compare their average amount of evaluation steps to our implementation of the NSGA-II algorithm.
The Cramér-von Mises test turned out to be the slowest of the applicable fitness metrics for the genetic algorithm, but was able to find more accurate solutions most

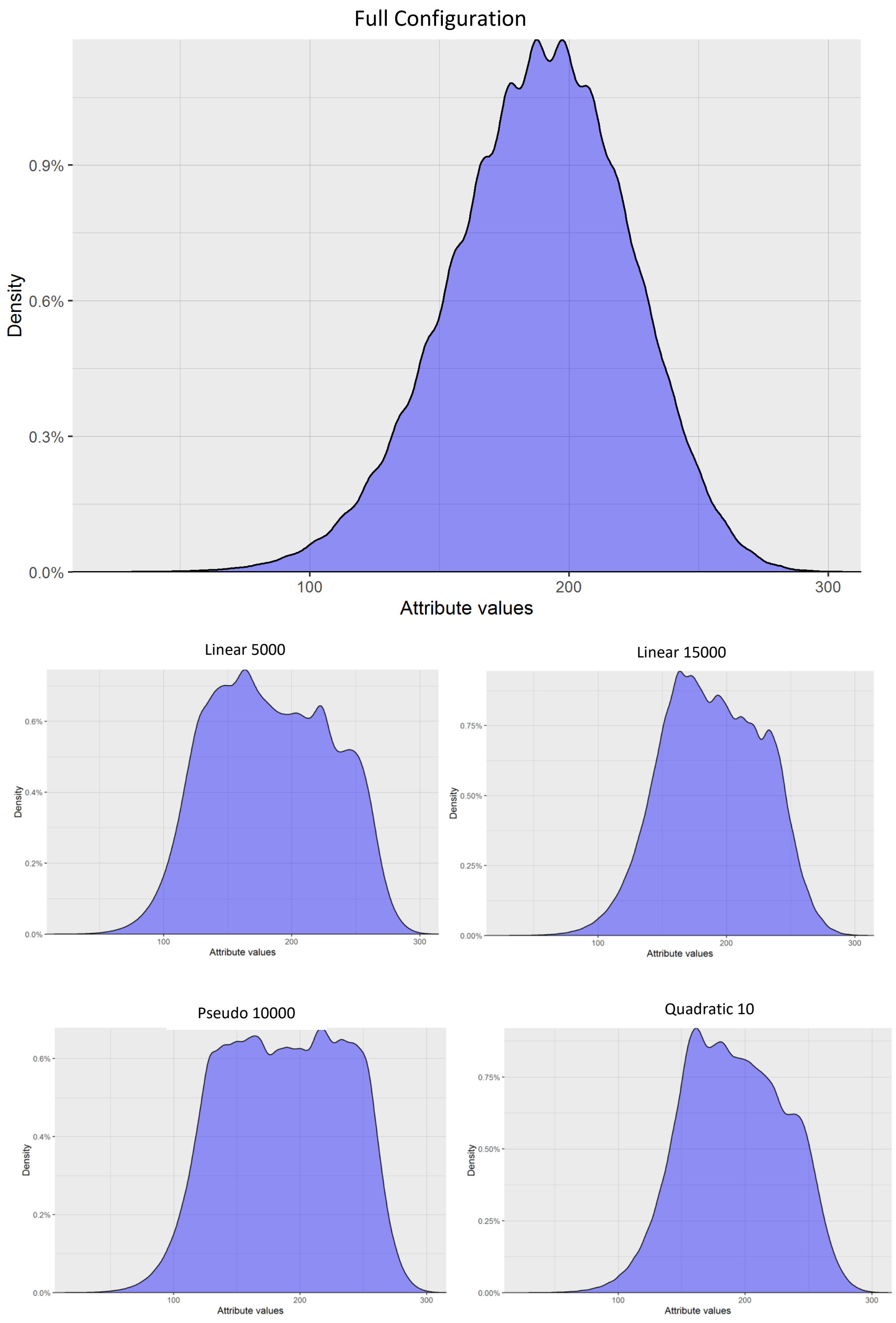


Figure 6.10: Variant distribution results of different sampling techniques in comparison to the fully configuration

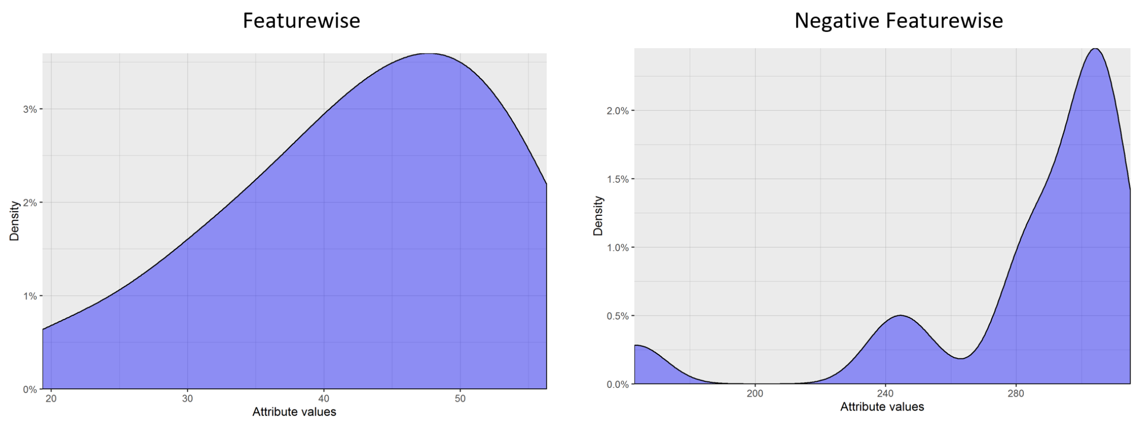


Figure 6.11: Featurewise and Negative Featurewise sampling heuristic

of the time. The fitness implementation is easy replaceable with other goodness-of fit techniques and we plan to add and evaluate other metrics such as the earth mover's distance (EMD) in future releases.
The biggest improvements in the accuracy of solutions might however be achieved with better variant sampling. The implemented heuristics are rather simple and static, and cannot react to any special characteristics of the variability model. Our main goal is to achieve a dynamic, adjustable heuristic, which tries to explore the possible amount of variants depending on prior knowledge and feedback of the variant generator. In future releases we want to add these feature either with a pre-computation step or with a dynamic heuristic.

7. Related Work

To our best knowledge the enhancement of large variability models to attributed realistic variability models with interactions, is not handled in any other publication yet. But as we mentioned in the literature study chapter, many publications are related to the generation of variability models, evolutionary algorithms, and the reasoning on attributes of variability models.
The first attributed variability model generator was presented by Segura et al. [SGB+12] within their BeTTy framework, which allows the creation of random variability models and attributed variability models. BeTTy allows the creation of integer attributes with a large amount of pre-defined distributions[1] found in the Apache Math package. They also implemented an evolutionary generator, to create hard variability models which maximize the memory consumption or CSP-solving time for the benchmark testing of analysis tools.
Mendonca et al.[MWC09] also provide a generator which is used at the SPLOT website. They generate variability models with cross-tree constraints in 3 CNF notation for the benchmark testing of analysis tools using SAT solvers. However they do not generate attributed variability models.
Segura et al. [SPH+14] use their ETHOM evolutionary algorithm for the generation of computationally hard feature models, to pose complex problems for the analysis tools of the community in comparison to purely random generated models. Genetic algorithms have recently gained more and more popularity for the analysis of attributed variability models. As we have shown in the literature study section they are widely used for the selection of optimal configurations under multiple constraints. For example, Ognjanovic et al.[OMG+12] use a genetic algorithm for the optimal feature selection with stakeholder constraints in business process management families (BPMF). Sayyad et al.[SIMA13b] combine static and evolutionary learning with an precomputed seed solution to find optimal configurations for large variability models with their Indicator-based evolutionary algorithm (IBEA). Pascual et al.[PLHP+15] compare different multi-objective evolutionary algorithms, for the dynamic reconfiguration of mobile applications under changing environment parameters such as

[1]http://commons.apache.org/proper/commons-math/userguide/distribution.html

battery status, available memory, or CPU load. Henard et al[HPHLT15] combine evolutionary algorithms with constraint solving, to calculate optimal configurations including cross-tree constraints on features and attributes.
We have not found any publications that use evolutionary algorithms for the creation of attributed variability models.

8. Conclusion

We performed a literature study on the current state of attributed variability. The survey indicated that the research community is missing on attributed variability models with realistic, real-world attribute values. Additionally interactions are widely ignored in the community. As the SPLConqueror dataset is, to our knowledge, the only publicly available dataset on real-world attribute measurements, we conducted an analysis on the prevalent attribute distributions. We identified four general patterns of distributions in the SPLConqueror dataset.
To overcome the lack of attributed variability models we built an generator for the creation of realistic attributed models including interactions. Its input are a non-attributed variability model with support of many popular formats, and a feature, interaction, and system variant distribution from the SPLConqueror dataset as target for a genetic algorithm's multi-objective optimization process. The result of the process is an attributed variability model with interactions, where the feature, interaction and system variant distribution is similar to the targeted input distributions. We support many adjustable parameters such as the system variant sampling, the fitness metric, early-stop criterion and weighted selection of results in the last pareto-front of the genetic algorithm. The implemented tool achieves a high scalability through a high degree of parallelization and we plan to improve it in further releases. Many components of the tool are modularized so that it is possible to adapt and extend the existing implementation.
We hope that our tool is able to help the research community to overcome the shortage of attributed variability models and enable them to test and evaluate their tools with our generated realistic attributed variability models.

9. Acknowledgments and Tool Availability

The tool and a quickstart guide is obtainable from Github at https://github.com/se-passau/thor-avm/tree/master/Thor. I want to express my gratitude to my supervisor Dr. Norbert Siegmund for the useful guidance, remarks and engagement throughout the process of this thesis. Furthermore I want to thank Dr. Stefan Sobernig especially for the help on statistical topics. Lastly I want to thank my supervisor Prof. Dr. Apel for allowing me to write this thesis.

A. Appendix

The implementation, the raw data and the evaluation results are provided digitally with this thesis.

Bibliography

[And62] T. W. Anderson. On the distribution of the two-sample Cramer-von Mises criterion. *The Annals of Mathematical Statistics*, 33(3):1148–1159, 09 1962. (cited on Page 16)

[BRN$^+$13] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, and A. Wąsowski. A survey of variability modeling in industrial practice. In *Proc. Int'l Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, pages 7:1–7:8. ACM, 2013. (cited on Page 1)

[BTRC05] D. Benavides, P. Trinidad, and A. Ruiz-Cortés. Automated reasoning on feature models. In *Proc. Int'l Conf. on Advanced Information Systems Engineering (CAiSE)*, pages 491–503. Springer, 2005. (cited on Page 6)

[CGR$^+$12] K. Czarnecki, P. Grünbacher, R. Rabiser, K. Schmid, and A. Wąsowski. Cool features and tough decisions: A comparison of variability modeling approaches. In *Proc. Int'l Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, pages 173–182. ACM, 2012. (cited on Page 1)

[DM97] D. Dasgupta and Z. Michalewicz. *Evolutionary Algorithms in Engineering Applications.* Springer, 1997. (cited on Page 10)

[DN11] J. J. Durillo and A. J. Nebro. jMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10):760–771, 2011. (cited on Page 41)

[DPAM02] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002. (cited on Page 41)

[HPHLT15] C. Henard, M. Papadakis, M. Harman, and Y. Le Traon. Combining multi-objective search and constraint solving for configuring large software product lines. In *Proc. Int'l Conf. Software Engineering (ICSE)*, pages 517–528. IEEE, 2015. (cited on Page 22 and 62)

[KCH$^+$90] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, DTIC Document, 1990. (cited on Page 1)

[Luk13] S. Luke. *Essentials of Metaheuristics.* Lulu, second edition, 2013. (cited on Page 12)

[LZ15] X. Lian and L. Zhang. Optimized feature selection towards functional and non-functional requirements in software product lines. In *Proc. Int'l Conf. Software Analysis, Evolution and Reengineering (SANER)*, pages 191–200. IEEE, 2015. (cited on Page 22)

[MG95] B. L. Miller and D. E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9(3):193–212, 1995. (cited on Page 12)

[MWC09] M. Mendonca, A. Wąsowski, and K. Czarnecki. SAT-based analysis of feature models is easy. In *Proc. Int'l Software Product Line Conference (SPLC)*, pages 231–240. ACM, 2009. (cited on Page 61)

[OMG$^+$12] I. Ognjanovic, B. Mohabbati, D. Gaevic, E. Bagheri, and M. Bokovic. A metaheuristic approach for the configuration of business process families. In *Proc. Int'l Conf. on Services Computing (SCC)*, pages 25–32. IEEE, 2012. (cited on Page 61)

[ORGC14] R. Olaechea, D. Rayside, J. Guo, and K. Czarnecki. Comparison of exact and approximate multi-objective optimization for software product lines. In *Proc. Int'l Software Product Line Conference (SPLC)*, pages 92–101, 2014. (cited on Page 22)

[PLHP$^+$15] G. G. Pascual, R. E. Lopez-Herrejon, M. Pinto, L. Fuentes, and A. Egyed. Applying multiobjective evolutionary algorithms to dynamic software product lines for reconfiguring mobile applications. *J. Systems and Software*, 103:392–411, 2015. (cited on Page 61)

[SA13] A.S. Sayyad and H. Ammar. Pareto-optimal search-based software engineering (POSBSE): A literature survey. In *Intl. Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*, pages 21–27. IEEE, 2013. (cited on Page 21)

[SGB$^+$12] S. Segura, J. A. Galindo, D. Benavides, J. A. Parejo, and A. Ruiz-Cortés. BeTTy: Benchmarking and testing on the automated analysis of feature models. In *Proc. Int'l Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, pages 63–71. ACM, 2012. (cited on Page 61)

[SGPMA13] A.S. Sayyad, K. Goseva-Popstojanova, T. Menzies, and H. Ammar. On parameter tuning in search based software engineering: A replicated empirical study. In *Intl. Workshop on Replication in Empirical Software Engineering Research (RESER)*, pages 84–90. IEEE, 2013. (cited on Page 21)

[She04] S. J. Sheather. Density estimation. *Statistical Science*, 19(4):588–597, 2004. (cited on Page 7)

[SIMA13a] A. S. Sayyad, J. Ingram, T. Menzies, and H. Ammar. Optimum feature selection in software product lines: Let your model and values guide your search. In *Intl. Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE)*, pages 22–27. IEEE, 2013. (cited on Page 21)

[SIMA13b] A. S. Sayyad, J. Ingram, T. Menzies, and H. Ammar. Scalable product line configuration: A straw to break the camel's back. In *Proc. Int'l Conf. Automated Software Engineering (ASE)*, pages 465–474. IEEE, 2013. (cited on Page 61)

[SKK+12] N. Siegmund, S. S. Kolesnikov, C. Kästner, S. Apel, D. Batory, M. Rosenmüller, and G. Saake. Predicting performance via automated feature-interaction detection. In *Proc. Int'l Conf. Software Engineering (ICSE)*, pages 167–177. IEEE, 2012. (cited on Page 7 and 22)

[SMA13] A.S. Sayyad, T. Menzies, and H. Ammar. On the value of user preferences in search-based software engineering: A case study in software product lines. In *Proc. Int'l Conf. Software Engineering (ICSE)*, pages 492–501. IEEE, 2013. (cited on Page 21)

[SPH+14] S. Segura, J. A. Parejo, R. M. Hierons, D. Benavides, and A. Ruiz-Cortés. Automated generation of computationally hard feature models using evolutionary algorithms. *Expert Systems with Applications*, 41(8):3975–3992, 2014. (cited on Page 61)

[SRK+12] N. Siegmund, M. Rosenmüller, M. Kuhlemann, C. Kästner, S. Apel, and G. Saake. SPL Conqueror: Toward optimization of non-functional properties in software product lines. *Software Quality Journal*, 20(3-4):487–517, 2012. (cited on Page 3)

[TXC+15] T. H. Tan, Y. Xue, M. Chen, J. Sun, Y. Liu, and J. S. Dong. Optimizing selection of competing features via feedback-directed evolutionary algorithms. In *Proc. Int'l Symposium on Software Testing and Analysis (ISSTA)*, pages 246–256. ACM, 2015. (cited on Page 22)

[WJ94] M. P. Wand and M. C. Jones. Multivariate plug-in bandwidth selection. *Computational Statistics*, 9(2):97–116, 1994. (cited on Page 8)

[ZYL14] G. Zhang, H. Ye, and Y. Lin. Quality attribute modeling and quality aware product configuration in software product lines. *Software Quality Journal*, 22(3):365–401, 2014. (cited on Page 22)

Eidesstattliche Erklärung:

Hiermit versichere ich an Eides statt, dass ich diese Masterarbeit 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 Masterarbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

Thomas Leutheusser

Passau, den 28. September 2016