University of Passau

Department of Informatics and Mathematics

Bachelor's Thesis

Generating Realistic non-functional Property Attributes for Feature Models

Author:

Philipp Eichhammer

June 27, 2014

Advisors:

Prof. Dr.-Ing. Sven Apel

Software Product-Line Group

Dr.-Ing. Norbert Siegmund

Software Product-Line Group

Alexander Grebhahn

Software Product-Line Group

Eichhammer, Philipp:
Generating Realistic non-functional Property Attributes for Feature Models
Bachelor's Thesis, University of Passau, 2014.

Abstract

In recent years the analysis of feature models, that contain non-functional properties, via analysis methods is rapidly gaining interest in the field of software product line engineering. Those methods are a useful and reliable tool, when testing is approved under real-world conditions. The majority of methods are, however, still tested with feature models containing random NFP values. Hence, we can not have confidence in the reliability under real-wold circumstances. In this thesis, we present a test suite, which provides the generation of test data, by offering a FM generator, which is capable of creating user adjustable FMs with real-world NFP values. Additionally, we also include feature interactions, which are not considered by other testing suites. Our approach is to use an evolutionary algorithm which optimizes randomly generated NFP values to result in a user selectable real-world NFP distribution. Thereby, testing of analysis methods on FMs with real-world based NFP distributions is made easy and comfortable.

Contents

List of Figures

List of Tables

List of Code Listings

1. Introduction

1.1 Motivation

Software Product Line (SPL) Engineering is becoming more and more important a means to develop reusable software due to its benefits in reuse of code and reduction of cost. The basic idea is to take advantage of the similarity among all variants of an SPL by generating different variants from a code base. For example, we can generate a variant of a database SPL with or without transaction support. Variants differ only in the user-selected features. This leads to the development of a basic implementation of an SPL and development of features that can be added to the basic implementation in order to create different variants. This avoids writing a new software program from scratch.
Since an SPL can contain multiple features that have constraints and dependencies, we need a model to specify all valid variants. The feature model (FM) is a common technique to define the variability of an SPL. It also offers a visual representation as a "tree", called feature diagram. A feature diagram has one root and all other nodes are features that can be/have to be selected (for a detailed description see Chapter 2).
New concepts entail new challenges, such as computing the optimum configuration with respect to certain properties of an SPL. In order to cope with those challenges different methods have been created to analyze FMs. In the course of this thesis, we specifically focus on analysis methods working on FMs containing non-functional Propertys (NFPs). These methods follow very different approaches, such as quality-aware analysis to check whether a configuration (with constraints) is possible [RFBRC+12] or, as another example, automated reasoning to extract information from a FM [BTRC05b].
Like every other software these analysis methods have to be tested, too. To do so either new testing tools, which automate the testing process, are developed or the testing is done manually. Both approaches use real and randomly generated FMs as an input, test the analysis method on this input, and compare the results with the expected ones. Although this is a reasonable approach, the problem is, that the potential of almost all feature models, they use, is not fully realized. In most cases

the complexity of the FMs is, at best, reduced to cross tree constraints, others even abstain from those. Feature interactions, an essential part of every FM and very important in respect to NFPs, are not taken into account at all. The main problem is, however, that the values of NFPs of FMs are generated randomly. In addition to the missing awareness of this problem, there is no relation to real-world NFP values. This means that we cannot have confidence in the analysis results. As a result, we test our analysis with artificial data sets ignoring interactions that may have a huge impact on the final result.
In [BTRC05b], the authors extracted information out of an FM using Constraint Programming. Based on this information, they computed the best product according to a criterion. Due to the lack of real-world measurement, they used artificial NFP values and did not include feature interactions. Hence, it is unclear whether their tool found the best product according to a criterion (eg. performance) only because the randomly generated NFP values result in a distribution of NFP values their tool could handle. That is, we do not know whether real distributions could lead to a problem that would not arise when testing with random values.
Unfortunately this is not an isolated example. There are a large number of papers dealing with (automated) analysis on FMs containing NFPs (e.g. [KAK09], *[...]*) and almost all of them use artificially and randomly generated NFP values. We conclude, there is a huge necessity for real-world-based NFP values in order to properly test analysis tools.

1.2 Goal of this Thesis

The goal of this thesis is to develop a testsuite for analysis methods that provides FMs with real-world based NFP distributions. The key component of this suite is an extended FM generator working on real-world datasets. Unlike any other generator, it also supports feature interactions based on actual distribution patterns (details in Chapter 3). Combining those features, interactions and assignment of real-world inspired NFP values, in one suite, we want to finally open the door to real-world based testing of analysis techniques.
To archive this goal, we first analyze real-world dataset in order to create a realistic NFP distribution (see Chapter 3), which our generator can later use. Next we extend the existing FM generator Betty [1] to enrich the generated model with feature interactions. We add feature interactions according to the number of distribution empirically found in real-world programs. Finally, we use an evolutionary algorithm to generate the NFP values for features and interactions based on a user selected NFP distribution and FM properties.

1.3 Structure of the Thesis

This thesis is structured as follows. In Chapter 2, we focus on the explanation of all definitions with the goal to provide a good basis for the understanding of the remaining thesis. In this context, we also perform the analysis of literature using FMs with NFPs as a bases for their testing. The following Chapter 3 covers the process of creating NFP distributions based on real-world data which later on will

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

be used as an input for the evolutionary algorithm explained in Chapter 4. Here the concept and the procedure of the algorithm will be explained in detail. Afterwards in Chapter 5, we perform the evaluation of the algorithm and show the correctness and performance of its components. The thesis concludes with the chapters about related work, future work and the final summary.

2. Background

This chapter covers and explains all the definitions needed for understanding the remaining parts of this thesis.

2.1 Software Product Line Engineering

Software produce line engineering (SPLE) is a special form of software engineering, with the aim of creating software product lines. A *software product line* (SPL) is a set of software systems, that belong to the same domain and only differ in features. The approach of SPLE is to use those similarities in order to build a set of similar software systems by reusing a shared set of features. The key idea is to implement a basis and some features in order to create software systems without having to start from scratch every time a new feature has to be added.

2.2 Feature Model

A *feature model* (FM) is a representation of all features, their relationships and constraints belonging to a software system, thus, describing its variability and all of its variants [CE00]. It can contain a feature diagram as a visual representation and some other information related to the SPL, such as descriptions of some features, stakeholder information and examples of possible systems.

2.3 Feature Diagram

A *feature diagram*[1] is a visual representation of a feature model. Its structure is similar to the data structure "tree" consisting of one root node, a set of nodes describing features of a system, a set of directed edges and a set of edge decorations (defined in [CE00]). In Figure 2.1 on the following page a simple feature diagram

[1] It is common that people talking about a feature model they actually mean a feature diagram because this is the main component of a feature model

of a phone SPL is shown. In this example the features are representing the nodes and the relationships are represented by the edges. In order to differentiate between different types of relationships (alternative, or) and features (mandatory, optional) the little filled or empty circles (on top of some of the features), the filled or empty parts of arcs (between the edges) and some logic statements are used. The circles indicate whether a feature is optional or mandatory and the arcs indicate in which type of relationship a feature is participating. In Figure 2.2 on the next page a legend containing all important information is drawn.
Concerning the different types of features, there are mandatory and optional features. *Mandatory features* are features that have to be part of every variant of the SPL. *Optional features* are features that can be part of a variant but do not have to be.
Concerning the different types of relationships, there are alternative and or relationships. When features are part of an *alternaive relationship*, only one of them can be selected but one has to be selected. When features are part of an *or relationship*, one or more (even all can be selected) of them can be selected but at least one has to be selected.
As an example, all phones (of this SPL) must have a processor and a camera but the NFC and 4G technology are optional. Following the example the processor according to the feature diagram either is an ARM or Snapdragon but cannot be both, since those are part of an alternative group. Additionally, we must select a camera (indicated by the filled circle) and at least a front or rear camera (indicated by the filled arc). We also can select both cameras.
We can also define cross tree constraints, which are given as a boolean formula. As an example, when the phone has 4G technology it also has to have a Snapdragon as the processor but must not contain the NFC technology.

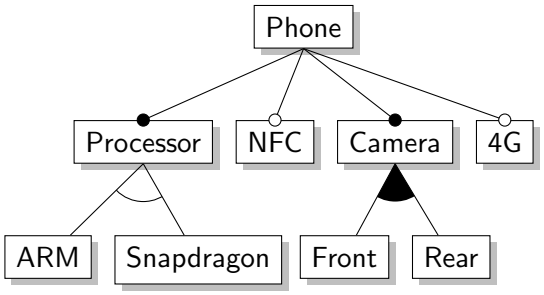


Figure 2.1: A feature diagram representing a phone product line.

2.4 Non-Functional Property

A *non-functional property* (NFP) is a property of a feature or a feature interaction, which does not affect the functional behavior of a variant. Those properties can be production costs or the amount of process power needed by this feature or interaction. In Figure 2.3 on the facing page the NFPs of the features of the phone SPL

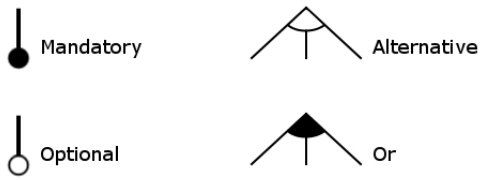


Figure 2.2: The legend of a feature diagram.

can be seen. When a phone only has a front camera, for example, the production cost are higher then when only having a rear camera.

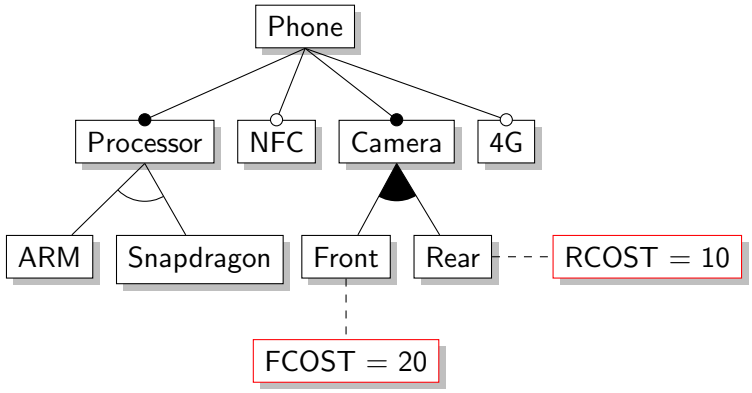


Figure 2.3: A feature diagram representing a phone product line containing non-functional Properties (marked by red borders). The costs for the feature ”Camera” compose of the costs of the front camera and the rear camera.

2.5 Feature Interaction

A *feature interaction* is a concept that appears when two or more features are selected at the same time. Two or more features are interacting with each other when their simultaneous selection leads to unexpected behavior [CKMRM03]. The unexpected behavior can be a performance drop or increased cost when two features are selected in combination. In Figure 2.4 on the next page we can see that the features *Snapdragon* and *Rear* interact with each other leading an additional energy consumption of 10 Joule and 20% more use of CPU capacity.

2.6 NFP Distribution

The *NFP distribution* of a feature model describes the number of variants having an equal NFP value. For example, we know how many variants share the same or equal performance. Such a distribution gives us a clue about the spread of performance values of a variable system as well as the number of performance optimal variants. Each point in the distribution refers to a single variant whose NFP values are defined by the sum of NFP values of its selected features and interactions.
It is important to note, that the NFP distribution used in this thesis refers to the distribution of NFP values of variants and not the NFP values of features or interactions.

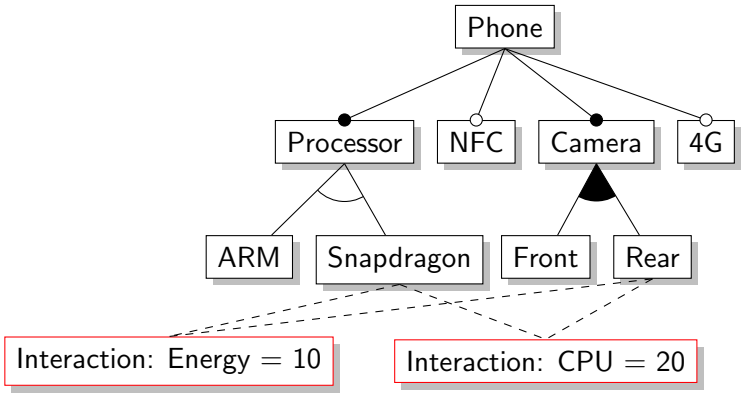


Figure 2.4: A feature diagram representing a phone product line containing a interaction.

As an example in Figure 2.5 most of the products have a performance value of 14.5 and the distribution is quite similar to a normal distribution. The creation and analysis of those distributions will be treated in depth in Chapter 3.

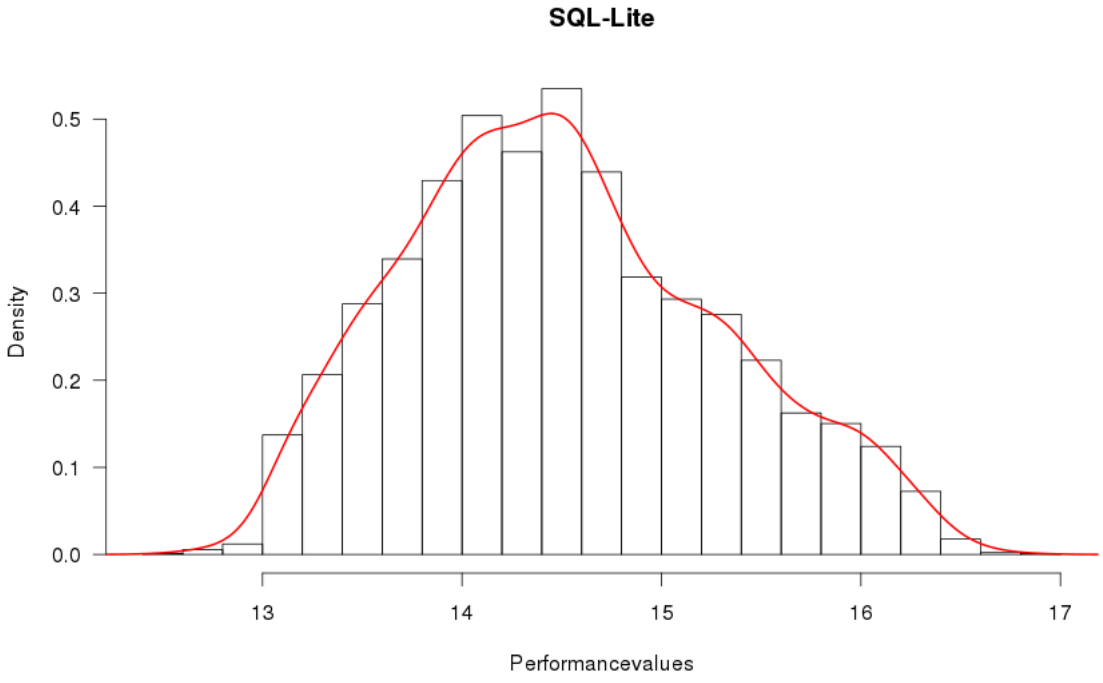


Figure 2.5: A NFP-Values distribution of the SPL "SQL-Lite".

2.6.1 Analysis of Literature

We analyzed the existing literature about analysis methods on FMs, with the aim of discovering the testing process of those methods. With our discoveries, we want to punctuate the necessity of our test suite.
The analyzed literature was found online, based on some catchwords. The resulting papers were: [Ale03, ABM+13, BMO09, BTRC05a, CdPL09, CPFD01, EMB07,

GWW+11, MARC13, RNMM08, RFBRC+11, SIMA13, Sol12, SRP, WDS09, WSNW]. Our suspicions of a testing process that included only random generated NFP distributions were confirmed by most of those papers. Unfortunately, 10 out of 16 papers did not include any description of their testing process, hence, we do not know their techniques but we assume that they also used random generated distribution. At least for the majority of them.
The remaining 6 papers did mention their testing techniques. Fife of them ([BTRC05a, RFBRC+11, SIMA13, WDS09, SRP]), however, tested with random distributions. Only one paper, in particular [Sol12], used an interesting approach of generating NFP distributions. Soltani et al., generated an NFP distribution by assigning NFP values to features that were produced by a random function with normal distribution. Depending on the number of variants and other factors, this distribution could result in a NFP distribution shaping a bell-shaped curve but this cannot be guaranteed. Nevertheless, this method can be counted as a testing technique using real-world based NFP distributions.
The five paper using random generated NFP distributions were not aware of the problems, such as the case of their analysis method failing on real-world NFP distributions, their testing techniques could lead to.
Concluding this analysis punctuated the necessity of a test suite for real-world NFP distributions.

2.7 Evolutionary Algorithm

An *evolutionary algorithm* (EA) is an algorithm orienting on the process of evolution with the aim of solving an optimization problem. It adapts its results to a specific environment step by step until the results are near the desired optimum [Mit97], [SHF+94].
The basic procedure is creating a population of chromosomes which are then evaluated, mutated, and crossed over to create a new population representing the next generation. Then, the same procedure is done with the next generation. This process is repeated until an exit condition is reached.
In detail, first an initial population is randomly generated. Afterwards, every single chromosome is evaluated using a fitness function. Based on this fitness value, each chromosome is checked whether it comes into the next generation. The selected chromosomes are then crossed over and mutated in order to build the next generation. This will be repeated until an exit condition, usually after a specific number of generations or a degradation of the overall fitness, is reached. Figure 2.6 visualizes this procedure.
The chromosomes represent the structure to be optimized. In our case, a chromosome is a set of NFP values as we describe in Chapter 4)

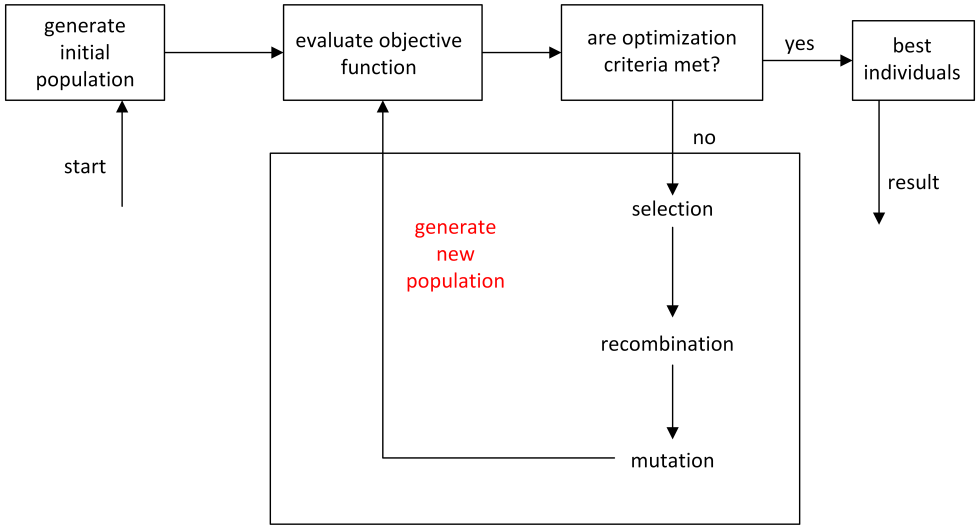


Figure 2.6: The procedure of an evolutionary algorithm.

3. Analysis of real-world Datasets

In this chapter, we focus, on the one hand, on the analysis of real-world datasets whose goal it is to determine typical NFP distributions of SPLs. On the other hand, the analysis of real-world NFP values and especially the relation between NFP values of features and interactions will be discussed. The results of this analysis form the basis of the behavior and evaluation of the algorithm.

3.1 Analysis of NFP Distributions

The analysis of NFP distributions lays one half of the groundwork for this thesis because thereby we know how real-world based distributions have to look like.

3.1.1 Origin of the data

We analyzed a diverse set of real-owrld SPLs ([SRK+11], [SKK+12], [SvRA13], [SRK+13]): Berkeley DB CE, Berkeley DB JE, SQLite, Apache, ZipMe, LinkedList, PKJab, LLVM, x264, Prevayler, EPL, Wget, AJStats, Email, h264, Elevator.
The measurement data for these systems consisted of the following NFPs: binary size, main memory consumption and performance whereas not all NFPs are available for all subject systems.

3.1.2 Analysis Results

As described in the beginning of this chapter the analysis of NFP distributions shows how real-world SPL's NFP values distribute over all variants. Those results can then be used for testing and optimizing the evolutionary algorithm generating an annotated FM and later on for evaluating the generated distributions.
After creating the NFP distributions we discovered that there are two main types of distributions: normal distribution and equal distribution. Within the category of normal distributions, we observed some subclasses: one bell-shaped curve (which can be described by one density function) and two or more bell-shaped curves (which

then are described by the maximum function of two or respectively more density functions). For a better understanding the following figures (Figure 3.1, Figure 3.2, Figure 3.3) each describe a NFP distribution shaping a one bell-shaped curve, two bell-shaped curve, and three bell-shaped curve. Furthermore Figure 3.4 shows an equal distribution which is best described by the density function of an equal distribution.

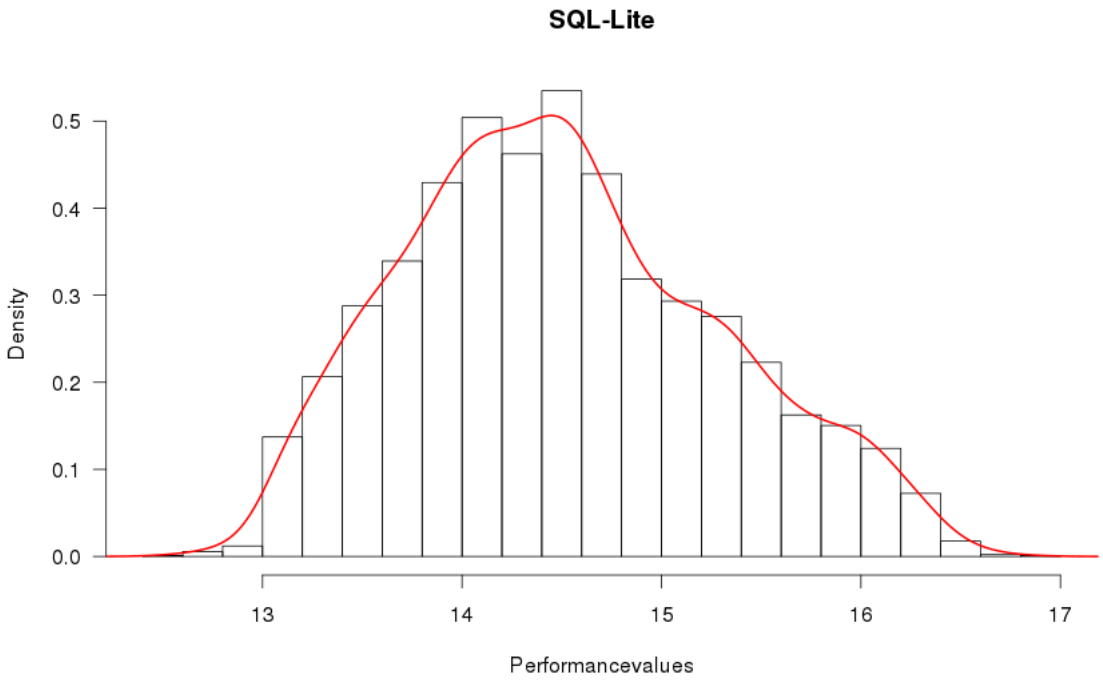


Figure 3.1: An NFP distribution of the SPL "SQL-Lite". We categorize this one as a one bell-shaped curve even though it is not perfectly shaped.

The analysis shows that 20 out of 23 distributions correlate to the type of a normal distribution whereas 9 of those have the shape of three bell-shaped curves. This makes the three bell-shaped curves the most common shape among the analyzed SPLs. The remaining 11 split into 8 two bell-shaped curves, 2 one bell-shaped curves and 1 distribution representing several bell-shaped curves. Three out of 23 distributions correlate to the density function of an equal distribution. We found that SPLs having a feature model with mostly optional features and no constraints usually result in a bell-shaped curve. Figure 3.5 shows the frequency of occurrence of each discovered shape.

3.1.3 Resulting Functions

Our goal is to extract the information of these distributions such that we can use them for generating similar distributions in a generated FM. Hence we need the distributions in a processable form. Therefor, we describe them as mathematical functions. To this end, we categorized them, as seen before, so it is easier to find the

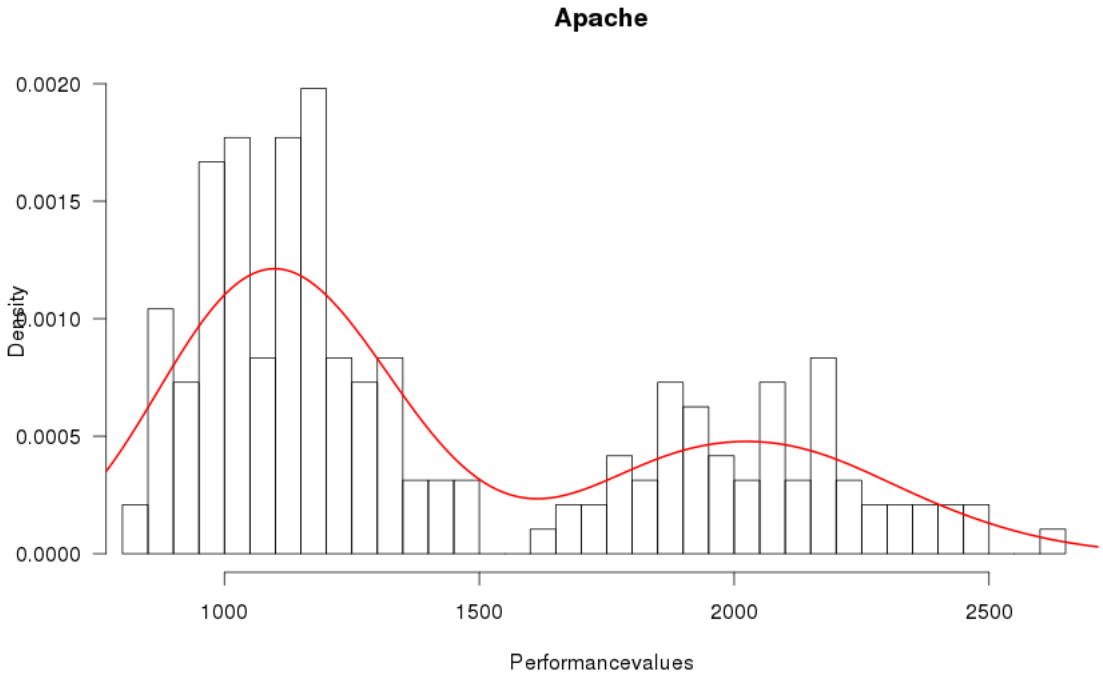


Figure 3.2: An NFP distribution of the SPL "Apache". We categorize this one as two bell-shaped curves for obvious reasons.

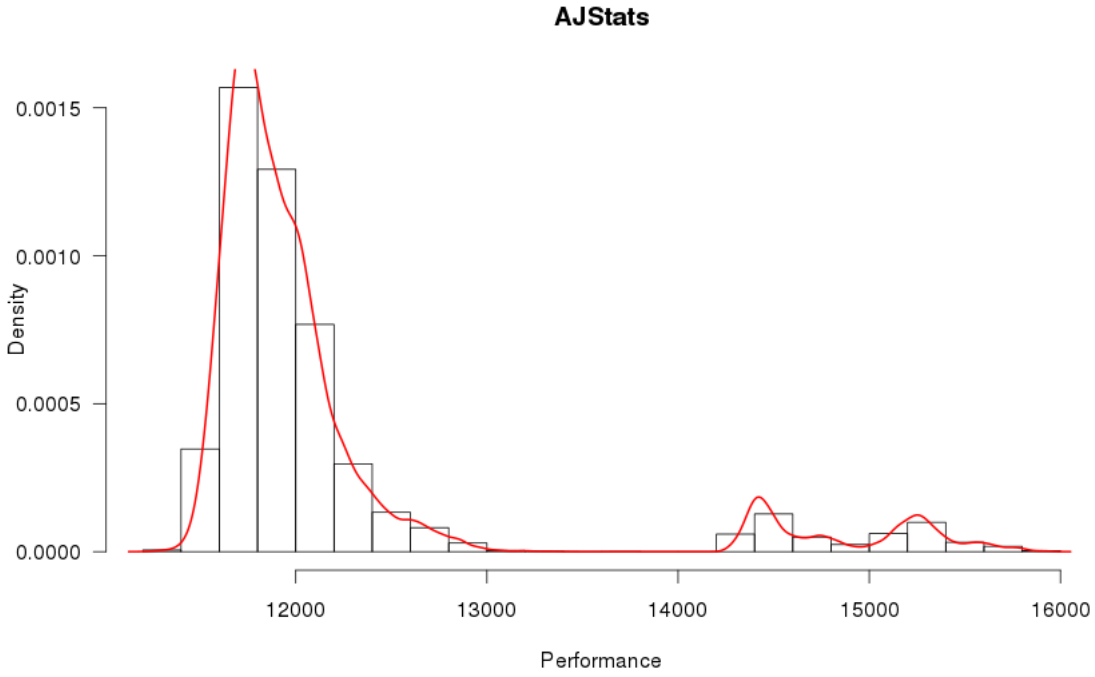


Figure 3.3: An NFP distribution of the SPL "AJStats". We categorize this one as three bell-shaped curves.

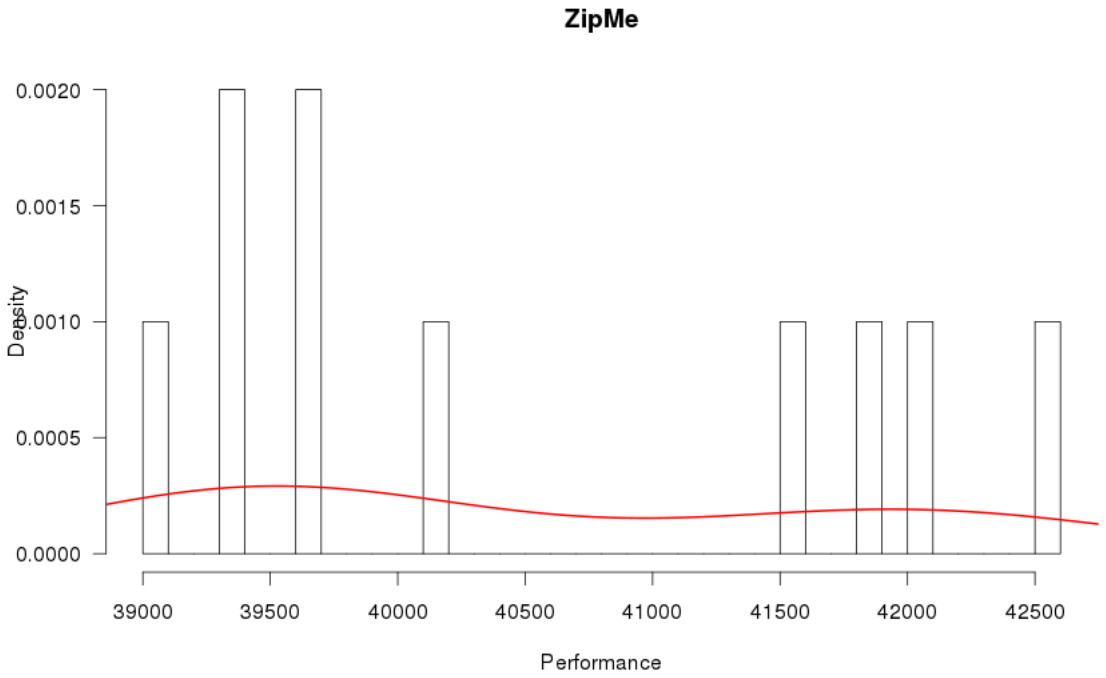


Figure 3.4: An NFP distribution of the SPL "ZipMe". We categorize this one as a equipartition even though there is a gap between some bars.

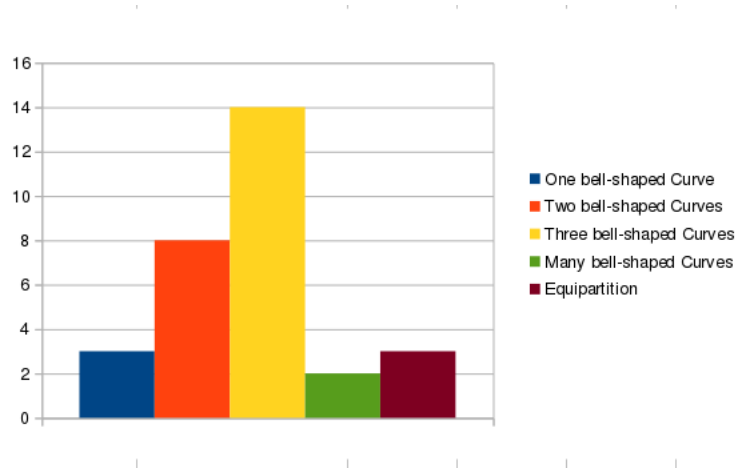


Figure 3.5: A bar chart showing the frequency of occurrence of each discovered shape.

functions describing their shapes. For example, the distribution shown in Figure 3.3 can be described by the maximum function of three density functions of the normal distribution which looks like the following one:

$$max((\frac{1}{\sqrt{2\pi\sigma_1^2}}\exp\{-\frac{(x-\mu_1)^2}{2\sigma_1^2}\}),(\frac{1}{\sqrt{2\pi\sigma_2^2}}\exp\{-\frac{(x-\mu_2)^2}{2\sigma_2^2}\}),(\frac{1}{\sqrt{2\pi\sigma_3^2}}\exp\{-\frac{(x-\mu_3)^2}{2\sigma_3^2}\}))$$

That way the algorithm has a point of reference when evaluating its generated distributions.
As can be noticed in the figures, most distributions do not have a perfect shape according to their classification. For example in Figure 3.3, the first curve (going from left to right) is relatively close to be shaped as a bell but the second and third differ substantially. So the function representing this distribution does not describe it as detailed as in the figure but that is not its purpose. The aim is to provide functions that can be used to create similar distributions on randomly generated FMs (for further information see Chapter 4).

Structure of the functions

The analysis shows that there are two types of distributions the SPLs can be described with. First the normal distribution and second the equal distribution.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\{-\frac{(x-\mu)^2}{2\sigma^2}\} \tag{3.1}$$

$$g(x) = \frac{1}{b-a}, a \leq x \leq b \tag{3.2}$$

Function (4.1) is the density function of the normal distribution containing two variables to adjust it. Those are σ^2 and μ at which σ^2 controls the gradient of the bell-shaped curve and μ the shifting on the x-axis. The bigger the number for σ^2 the flatter the gradient of the curve and vice versa. If μ is a positive number then the curve will be shifted into the positive x-axis and vice versa. The following figures (Figure 3.6 and Figure 3.7) visualize the differences.

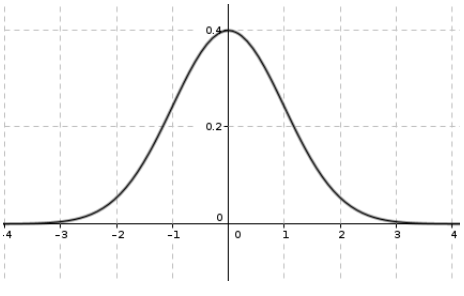


Figure 3.6: f(x) with $\sigma^2 = 1$ and $\mu = 0$

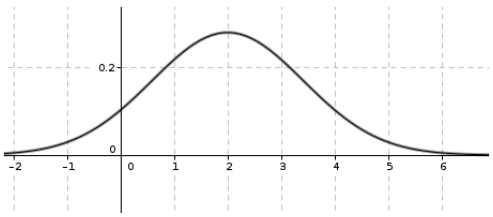


Figure 3.7: f(x) with $\sigma^2 = 2$ and $\mu = 2$

The important characteristic of Equation 3.2 is that it describes a constant function. With constant, we mean that every constant function will do due to the fact that the algorithm only orientates on the locale growth. (More in Chapter 4)
With these two functions, we can describe all discovered real-world distributions.

3.2 Analysis of NFP Values

In the first part of this chapter, we focused on how to determine and recreate NFP distributions of existing SPLs. This part focuses on the relationship between NFP values of features and NFP values of feature interactions. This analysis is important in order to satisfy our requirement of our extended FM generator to produce "real-world" FMs.

3.2.1 Analysis Results

Surprisingly, there is a large difference between the NFP values of features and those of interactions. The interactions tend to have a limited impact on NFPs and are not wide-spread on the range of values as features are. The distributions indicate that they are normal distributed. Although features are also concentrated around a zero value (i.e. no influence on the NFP), they are more widely spread. Additionally the concentration is not as strong as it is the case with the interactions. The following figures (Figure 3.8, Figure 3.9) give a insight into the different NFP values.

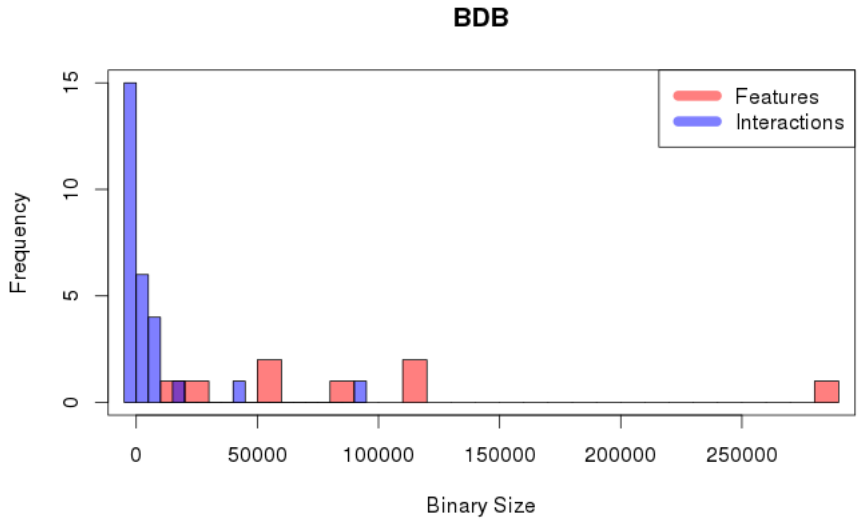


Figure 3.8: The NFP values of the SPL "BDB"

In further research we discovered a distribution of the NFP values of interactions. As Section 3.2.1 shows, about 50 percent of NFP values of features have a value of zero. The remaining 50 percent split into 15% of the values contained in the interval $]0, 0.3 \cdot z]$ with z being the highest NFP value of both features and interactions. 25% of the values contained in the interval $]0.3 \cdot z, 0.7 \cdot z]$ and 10% within the interval $]0.7 \cdot z, z]$.

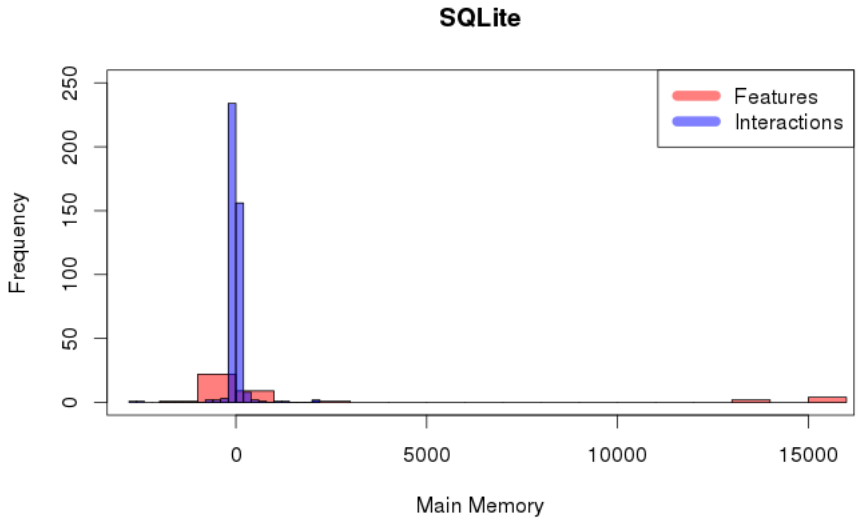


Figure 3.9: The NFP values of the SPL "SQlite"

% of values	interval
50%	0
15%	$]0, 0.3 \cdot z]$
25%	$]0.3 \cdot z, 0.7 \cdot z]$
10%	$]0.7 \cdot z, z]$

Table 3.1: This table shows the distribution of NFP values of the examined interactions. z is the highest NFP value of both features and interactions.

These results influence the approach of the generator when assigning the NFP values of the interactions by using the same distribution. (More detail in Chapter 4)

4. Algorithm

In this chapter, we focus on the structure and procedure of the algorithm used to create a FM whose NFP distribution is similar to a user-selected realistic one.

Basic Definitions

The following terms are used throughout this chapter, thus, it is important to know their definitions:

- A *population* is a set of chromosomes.
- A *chromosome* (or sometimes called *individual*) is a representation of the object that is optimized. In our case a chromosome is a set of NFP values of a FM.
- A *gene* is a single part of a chromosome containing a value that can be changed. In other words, a chromosome is a set of genes. In our case a gene is a single NFP value of a feature or interaction.

Figure 4.1 visualizes those correlations.

The following mathematical structures apply for this chapter:

- C is the set containing all Chromosomes and $n = |C|$ is the number of chromosomes contained in C
- G_i is the set containing all Genes of a chromosome $c_i \in C$ and $i \in \{1, ..., |C|\}$
- $f : C \to \mathbb{R}$ represents the fitness function assigning each chromosome an individual fitness value
- $p : C \to [0, 1]$ with $[0, 1] \in \mathbb{R}$ represents the function assigning each chromosome its probability of survival

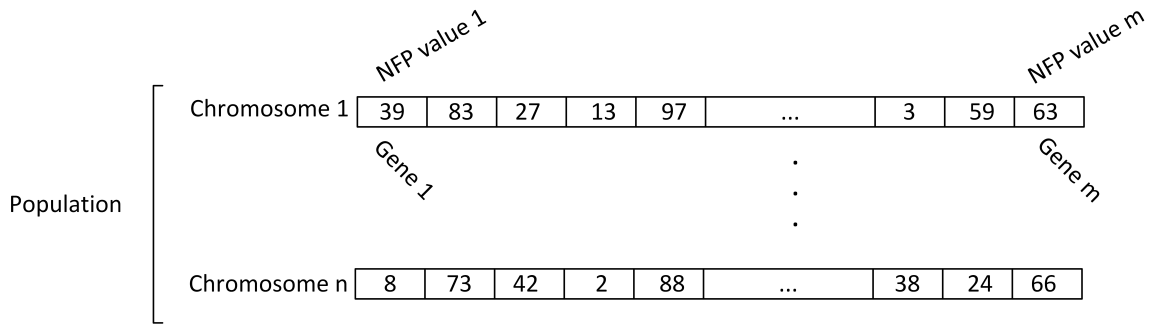


Figure 4.1: The structure of the evolutionary algorithm, we use. As explained, the population is a set of chromosomes and each chromosome is a set of genes. In our case the genes are NFP values belonging to features and interactions.

4.1 Structure of the Generator

This section gives a brief introduction into the structure of the whole generator containing the algorithm. This is helpful for getting a better understanding of the process of generating an FM.
The general process is following. The user specifies the basic parameters of the FM, such as number of features, probability of a feature being mandatory or optional, probability of a feature being part of an alternative or a or relation, percentage of cross tree constraints and interactions. These input values are used to generate a feature model using BeTTy [SGB+12]. The generated FM is then handed over to the SAT solver which is checking on its validity and calculates possible variants. The resulting model, when valid, is afterwards handed over to the algorithm optimizing the FMs NFP distribution to the given target function.
The above mentioned process consists of three main parts. One part is the algorithm which will be explained in detail in the next sections. The other two parts are, on the one hand, the SAT solver and, on the other hand, the FM generator.

Feature Model Generator

Starting with the ”BeTTy” FM generator ([SGB+12]), its basic function is to generate a FM containing mandatory and optional features, cross tree constraints and AND and OR relationships between features. We extended this process by additionally generating feature interactions. All those characteristics can be adjusted by the user, so the FM fits his needs. For example, the user can create a FM consisting of 20 features at which the probability of each one of them to be optional is 70 percent. Additionally 28 percent of the features are part of a cross tree constraint and 30 percent interact with other features.
On top of ”BeTTy”, we developed the functionality of creating feature interactions of first (two features interact), second (three features interact) and third order (four features interact) whose distribution also can be configured.

SAT Solver

The second part of the generator is a SAT solver whose task it is to evaluate the generated FM in terms of validity and the ability to find possible variants. To this end, we use the popular "Sat4J" SAT solver.[1] The procedure is as follows. First, we transform the generated FM into a Boolean expression then check it for validity and afterwards a particular number of possible variants is generated. As the number of possible variants increases exponentially with the number of features, we limit the solver to calculate a maximum amount of 5000 variants. In order to still be able to create a very similar distribution as the FM would have with all of its variants, we select some characteristic variants (e.g., the maximum variant and the minimum variant) and some random variants. We give more detail in Chapter 5.

4.2 Procedure of the Algorithm

The algorithm used to create FMs with adjustable NFP distributions is an evolutionary algorithm converging the NFP distribution of the FM towards a given target function. In this section, we focus on the specific behavior of this algorithm.

4.2.1 Input

The input consists of one of the distribution functions Equation 3.1 or Equation 3.2, an interval and the values for the following properties: The range of NFP values of the generated NFP distribution that are counted as one group (called steps), the number of chromosomes of the population, the accuracy of the resulting function, the maximal number of generations and whether the given function is the density function of the normal distribution or the one of the equal distribution.
The interval defines the area of the real numbers, on which the given function shall be used on as the target function. This function is used later on for evaluating the generated NFP distributions. The maximal number of generations is used by an exit condition. The remaining values are explained later on.
Regarding the function, however, it has to have some constraints. First the function neither must be zero nor below zero for any values of the given interval, second it has to be defined on a single coherent interval and third it has to be one-dimensional.

4.2.2 Procedure

This section gives a brief overview of the procedure of the algorithm.
When starting the algorithm, an initial population containing a specific number of chromosomes is created. A chromosome is a set of NFP values allocated to the features or interactions of the generated FM. Next, we compute a fitness value for each chromosome using a fitness function described later. Then, the selection process selects the chromosomes using a random number and the probability of survival to select the chromosomes of the next generation. Afterwards, those chromosomes are manipulated by crossover and mutation. This process is repeated until the exit condition is reached. This procedure is visualized in Figure 2.6

[1] http://www.sat4j.org/

Initial Population

The initial population is generating the initial NFP values of features completely randomly using a random function with equal distribution and choosing from values of the interval $I = [1, 100] \in \mathbb{N}$. The NFP values of the interactions are, however, tailored to a special distribution, we discovered in the course of this thesis. (More information in Chapter 3).

Fitness function

The fitness function is one critical part of the algorithm due to its function of evaluating the chromosomes. A good evaluation lays the foundation of a good selection and fast and accurate optimization. Therefore a poor evaluation is very insufficient and counterproductive.
In our case, the fitness function has to compare the generated NFP distribution with the given target function in order to allocate a fitness value. Additionally, we implemented three optimization strategies, which especially focus on characteristic parts of the given function and assess these parts particularly heavy. Those are meant to support the fitness function. A detailed description of those optimizations and their performance enhancements is given later in Chapter 5.
The comparison of the generated NFP distribution and the given target function is, however, challenging as the distance of their resulting values are not directly comparable. In Chapter 3, we explained, that the real-world NFP distributions can be best described by the density functions of the normal distribution and the equal distribution. The problem with the function of the normal distribution is, however, the fact that their return values are mostly between zero and one. At the same time, the *values* of the key-value pairs of the generated NFP distributions are natural numbers (e.g. 20, 60, etc.). In order to compare both functions, we cannot calculate the distance between those, but are forced to calculate the distance between their local growth. As the number of *keys* of the key-value pairs are discrete, there is only a discrete number of different values, where we can compare the local growth.
In order to calculate this local growth, we need to do some preparation. First, we take the number of keys of the key-value pair and divide the length of the given interval $I = [a, b] \in \mathbb{R}$ of the function by this number. The resulting value is the size of the steps, we have to go within the interval to get to the next result value, at which we can compare the function and the NFP distribution again. The function l then can return the x values of the function at which we can compare with the generated NFP distribution. k is the number of keys.

$$steps = \frac{|I|}{k} \tag{4.1}$$

$$l(i) = \begin{cases} \sum_{j=0}^{i-1} l(j) + steps, i \neq 1 \\ a, i = 1 \end{cases} , i \in \{1, ..., k-1\} \tag{4.2}$$

Finally, we can calculate the local growth of the given function by using Equation 4.3. The local growth of the generated NFP distribution is calculated by using Equation 4.4.

$$g(i) = \frac{l(i)}{l(i-1)} \tag{4.3}$$

$$h_p(i) = \frac{c_p(i)}{c_p(i-1)} \tag{4.4}$$

$$i \in \{2, ..., k\} \tag{4.5}$$

$c_p(i)$ is returning the value belonging to the key i of the key-value pair of the chromosome c_p with $p \in \{1, ..., |C|\}$.
In the special case, when $c_p(i-1) = 0$, we replace the 0 with a 1. This is needed by Equation 4.4, as a division by zero is not allowed. However, this does not influence the local growth, as either the following value $c_p(i)$ is also zero or not zero. In case it is zero, the resulting growth is zero, which is correct. In case it is not zero, the correct growth is the value of $c_p(i)$.
After the steps are calculated, the fitness function can calculate the fitness values. This is done using the following formulas:

$$f(c_p) = m - d(h_p, g) \tag{4.6}$$

$$m = max(d(h_j, g)), \forall j \in \{1, ..., |C|\} \tag{4.7}$$

$$d(h_p, g) = \sqrt{(h_p(2) - g(2))^2 + ... + (h_p(k) - g(k)))^2} \tag{4.8}$$

Here, f is the fitness function returning the fitness value of the chromosome c_p, and d is the metric, we use for calculating the distance between the local growth of the given target function g and the one of the generated NFP distribution h_p. Furthermore, d is the metric induced by the 2-norm.
After calculating the deviation of the NFP distribution from the target function, the fitness value is calculated by subtracting the deviation from the highest deviation of any chromosome. Thus, we ensure that the NFP distribution with the lowest deviation gets the highest fitness value, which is important for the probability of survival.
Closing, the probability of survival, essential for the selection process, is calculated using the following formula:

$$p(c_i) = \frac{f(c_i)}{\sum_{j=1}^{n} f(c_j)} \tag{4.9}$$

Where p is the function assigning the probability of survival to a chromosome $c_i \in C$ with $i \in \{1, ..., |C|\}$.

Selection

After evaluating each chromosome using the fitness function, we start the selection process. The aim is to select the chromosomes for generating the next generation.

To do so, it randomly picks the chromosomes considering the probability of survival. Thereby the probability of survival directly affects the probability of a chromosome to be picked. This means in our case, that a chromosome with a high fitness value is more likely to be picked than a chromosome with a low fitness value. As indicated in Equation 4.9 the sum over all probabilities of survival is 1 such that every chromosome c_i is assigned an interval $I_i \subset [0,1] \subset \mathbb{R}$ with $i \in \{2, ..., |C|\}$ and:

$$I_i =]a_i, a_i + p(c_i)] \tag{4.10}$$

$$a_i = \sum_{j=1}^{i-1} p(c_j) \tag{4.11}$$

Whereas a_i is the sum of all probabilities of survival of the previous chromosomes and I_1 is a special case due to the inclusion of the lower border of the interval, which is the number zero: $I_1 = [0, p(c_1)]$.
After assigning the intervals, a random number $z \in [0,1] \in \mathbb{R}$ is selected. This number determines the chromosome to be picked for the next generation, depending on which interval it is an element from. For example the chromosome c_k is picked, when $z \in I_k$ with $k \in \{1, ..., |C|\}$. This process (selecting z and choosing the correct chromosome) is repeated $n = |C|$ times whereas one chromosome can be selected several times. Figure 4.2 is visualizing the selection process.

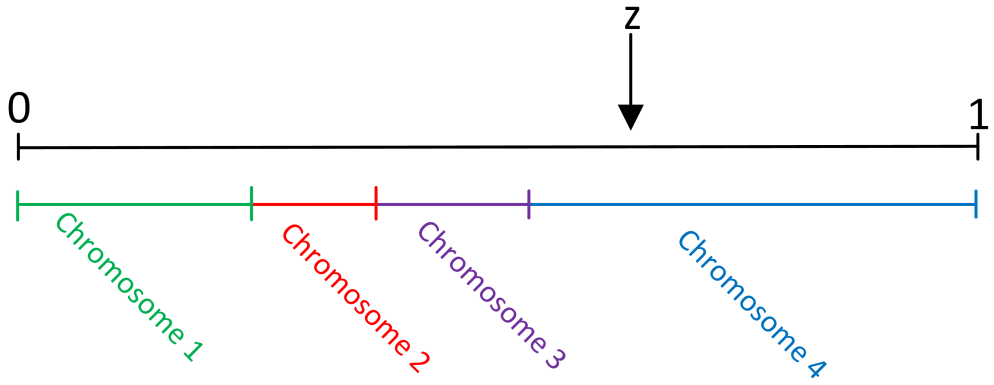


Figure 4.2: The selection process. z is the random number. The different colored lines represent the intervals I of the chromosomes 1 to 4. Chromosome 4 has the highest fitness value, thus, the highest probability to be picked.

Crossover

As mentioned earlier, we use crossover as a method for manipulating a two chromosomes of the next generation. This process combines a pair of chromosomes in order to get two new ones, consisting of a mixture of their NFP values. In our implementation, we used the method of *single-point crossover*. Thereby, two chromosomes $c_i, c_j \in C$ with $c_i \neq c_j$ are picked randomly and are crossed over by the probability of $p_c = \frac{1}{|C|}$. Every chromosome occurs only once in a pair. After a pair was selected and the probability of crossover occurs, a random number $z \in \{1, ..., m\}$, where $m = |G_i| = |G_j|$, is chosen. This number determines

a gene of c_i where all following $g_i \in G'_i = \{g_{i_z}, ..., g_{i_m}\} \subseteq G_i$ are swapped with $G'_j = \{g_{j_z}, ..., g_{j_m}\} \subseteq G_j$. The resulting new chromosomes have the following sets of genes: $G_j = \{g_{j_1}, ..., g_{j_{z-1}}, g_{i_z}, ..., g_{i_m}\}$ and $G_i = \{g_{i_1}, ..., g_{i_{z-1}}, g_{j_z}, ..., g_{j_m}\}$. Figure 4.3 visualizes that scenario. Does the probability of crossover not occur, then both chromosomes are part of the next generation without being crossed over.

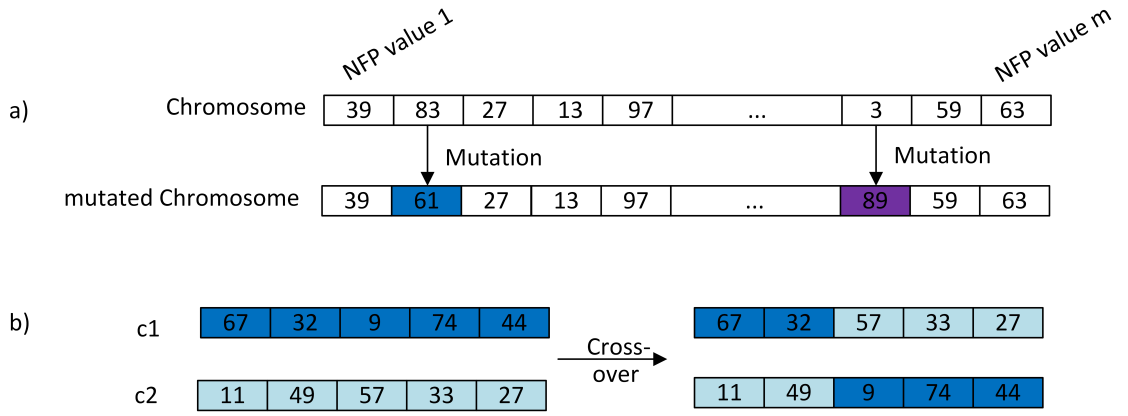


Figure 4.3: a) The *point mutation* of genes of a chromosome. Thereby, single NFP values are replaced by randomly generated new ones. b) The *single-point crossover* process.

Mutation

Another method for optimization is *point mutation*, which mutates single genes of a chromosome. The procedure is following. Every gene of a chromosome c_i is taken and mutated with the probability $p = \frac{1}{|G_i|}$ for $c_i \in C$. In our case mutation means to replace the existing NFP value of a feature or interaction by a randomly generated new one. This kind of mutation is called "point mutation" and Figure 4.3 shows this process.
There are other kinds of mutation, that are not used in this work but treated in Chapter 8.

Exit condition

As the evolutionary algorithm optimizes the NFP values of a FM to a certain target function, and optimization means, in this case, approximation, it probably never fully reaches this function and would run forever. So in order to terminate it has to know at which level of approximation it is done calculating. This information is given in the form of exit conditions, that are checked for fulfillment, after each generation. When one of them is fulfilled, then the algorithm will be terminated.
In our implementation the program will be terminated, when one of the following conditions are fulfilled. This is the case, when either a user adjustable number of generations is reached or a chromosome has reached a user adjustable minimal difference to the target function.
For example, we have a population of chromosomes, the maximal number of generations is 2000, the minimal distance is set to 1.0. The execution of the algorithm

will be terminated, when a one of the above criteria is reached.

4.2.3 Output

When the evolutionary algorithm is terminated, the best global chromosome is picked and its data, in our case the NFP values, is marked as the optimum result of the algorithm. In our case the NFP values are stored in the final FM file and is ready to be tested.

5. Evaluation

In this chapter, we focus on the evaluation of the evolutionary algorithm and the SAT sampling in order to see the performance enhancement of the optimization strategies and other components of the algorithm. Additionally, we evaluate the accuracy of the SAT sampling.

5.1 Evolutionary Algorithm

When evaluating an evolutionary algorithm its fitness function is the most important part to evaluate on. However, there are other components that also influence the performance of it. In the following sections, we focus on the performance enhancements of the fitness function, the quantity of population and the exit condition.

5.1.1 Fitness Function

The fitness function is the essential part of an evolutionary algorithm due to its task of evaluating the generated distribution relative to the given target function. Hence, its quality directly affects the speed and accuracy of the algorithm. In this section, we focus on different optimization approaches of the fitness function that where tested during the course of this thesis.

Metric

As can be seen in Chapter 4 the fitness function compares the generated distribution with the given target function respective to their distance. In order to measure the distance between two functions a metric is used. As the generated distribution, however, is not described by a continuous function but by a discrete number of key-value pairs, with the *key* being the x value and the *value* being the y value, the metric can not include an integral. Instead we chose to use the popular metric created by the Euclidean norm.

$$d(a,b) = \sqrt{(a_1 - b_1)^2 + ... + (a_n - b_n)^2} \tag{5.1}$$

Equation 5.1, also known as the Pythagorean theorem, is used in our implementation for measuring the distance between a and b. In our case a and b are the local growths of the given function and a generated NFP distribution. The Euclidean metric is very suited for this task due to the fact that it provides the best evaluation we tested. The other norms, we tested, were the 1-norm, several other p-norms and the maximum norm. The maximum norm was not very suited due to just delivering the highest aberration. The 1-norm was not as accurate as the 2-norm and the other higher p-norms delivered a higher accuracy but also needed more execution time. As a compromise between time and accuracy, we chose the 2-norm.

Optimization Strategies

The fitness function, comparing the growth of a distribution and the given target function in order to assign fitness values, has a big performance potential with the goal of speeding up the algorithm. In order to achieve this speed up, we implemented the following three optimization strategies, that support the fitness function. It is worth to mention, that all following strategies use a system of punishment and/or reward.
By the first strategy, we realized a system of punishment and reward where we compare the distribution and function by their local growth and punish the distributions, when growing positive whereat the function is growing negative and vice versa. Additionally we reward distributions that have the same growth as the given function and simultaneously a small difference in growth. For example consider the two generated distributions and the given function seen in Figure 5.1 and Figure 5.2. The first distribution has the shape of a one bell-shaped curve whereas the second one and the function have the shape of a graph consisting of two bell-shaped curves. Considering a fitness function without this optimization, the first distribution would get a higher fitness value than the second one due to the fact that the average local growth of the first distribution is more similar to the target function then the average local growth of the second one. When using this optimization, however, the second distribution gets a higher fitness value instead due to the punishment of the first distribution. This leads to a more precise selection and, thus, less execution time of the algorithm. During our tests we noticed, that this optimization is the fastest one, throughout the range of different target functions, compared to the other two when being the only one activated. The reason for this is that this process is the most precise in punishment and reward due to directly intervene when a aberration is noticeable.
For the second strategy (a system of punishment), we discovered that the random generated initial population is most likely to differ form the given function in several areas but the biggest aberration is centered in the middle of the interval, the given function is defined on. So a different weighting of these areas is boosting the accuracy and speed of the algorithm. To antagonize this circumstance we implemented a system of punishment where a big aberration in the middle is more punished than a

big aberration at the ends. For example there are two generated NFP distributions and a given function defined on the interval $[0, 10]$. The first distribution has a big aberration in the middle of the interval (about 5) and the second one a small aberration. But the second one has a big aberration at the ends of the interval (0 and 10). In a system without weighting the first distribution could be favored due to its low aberration at the ends. This would lead to a wrong selection and thereby to a longer execution time due to the fact, that there are lower quality distributions in the next generation. The reason for the unimportance of the aberration at the ends is that most distributions are very similar, means very low concerning the values, so the aberration is very likely to shrink in the following generations. This strategy is the second fastest, as it also intervenes mutiple times but only affects the inner aberrations.
The third optimization is a system of reward, checking for the change in growth of both, the generated and given functions. The change of growth, interesting for this strategy, is the transition form raise to fall and vice versa. These changes are counted and afterwards compared for equality. When the changes of growth of the generated and the given function are equal the fitness value of this chromosome is increased. When implemented isolated it has the slowest execution time due to only rewarding the proper distributions. In case all distributions have the same number of changes in growth as the given function then this strategy is no longer rewarding any chromosome, thus the effect is gone.
It is important to note, that the third strategy is only effective when used with a normal distributed target function due to the fact that generated NFP distributions often have less variants and therefore more gapes in between the single bars. This effect can be seen in Figure 3.4. Thus, there are often changes in growth, which the equal distributed target function does not have. Additionally it is worth to mention, that all punishment and reward factors, those strategies use to change the fitness value, are chosen based on testing throughout the development process.
When combining pairs of these optimization strategies (e.g. the first with the third one) the speed up tends to go higher but there is not a huge difference.
When combining all three of them, which is only possible with normal distributed target functions, however, the gain in performance is very impressive. They complement each other as the third strategy assorts the proper distributions and the other two support the fitness function. This configuration reaches a speed up of up to 3.
Table 5.1 shows all possible combinations of the different strategies and their speed up compared to the fitness function without any optimizations.
The following FM were tested. The tests of them all were tested repeated 10 times:

- First FM with 20 features, 902 variants, 10% CTC, 10% Interactions, probability of a feature being:
  - optional: 70%
  - mandatory: 10%
  - part of alternative relation: 10%
  - part of or relation: 10%

  Accuracy of resulting function: 6.0 and target function:
  $f(x) = max((1/(2 * 3.14)) * (exp(-(x-4)^2/(2))), (1/(2 * 3.14)) * (exp(-(x -$

$8)^2/(2))))$
There were 100 chromosomes in the population.

- Second FM with 30 features, 5000 variants, 10% CTC, 10% Interactions, probability of a feature being:
  - optional: 50%
  - mandatory: 20%
  - part of alternative relation: 10%
  - part of or relation: 20%

  Accuracy of resulting function: 2.0 and target function:
  $f(x) = max((1/(2 * 3.14)) * (exp(-(x-4)^2/(2))), (1/(2 * 3.14)) * (exp(-(x-8)^2/(2))))$
  There were 100 chromosomes in the population.

- Third FM with 20 features, 24 variants, 30% CTC, 10% Interactions, probability of a feature being:
  - optional: 10%
  - mandatory: 70%
  - part of alternative relation: 10%
  - part of or relation: 10%

  Accuracy of resulting function: 2.0 and target function:
  $f(x) = 1$
  There were 100 chromosomes in the population.

- Fourth FM with 20 features, 5000 variants, 10% CTC, 10% Interactions, probability of a feature being:
  - optional: 70%
  - mandatory: 10%
  - part of alternative relation: 10%
  - part of or relation: 10%

  Accuracy of resulting function: 2.0 and target function:
  $f(x) = 1$
  There were 100 chromosomes in the population.

Granularity of NFP Distribution

As mentioned above the data of the NFP distribution is not a function but a discrete number of key-value pairs that are created summing up the configurations having the same product NFP values. The problem, however, is the fact that just summing up the variants, having the exact same NFP values, would lead to many key-value pairs with one variant. As seen in Figure 5.3 the resulting distribution would not

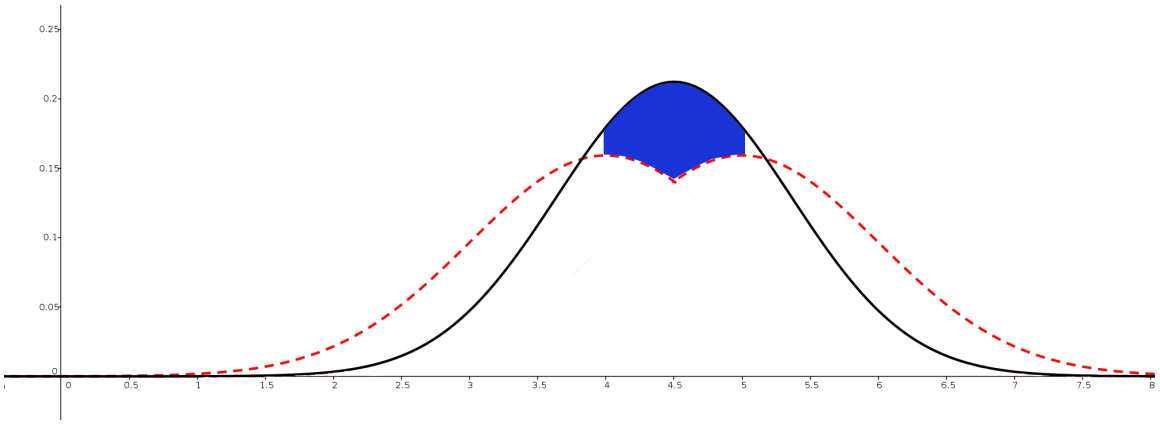


Figure 5.1: The first distribution having a better average growth then the second one but a wrong direction of growth between the x values of 4 and 5. Marked by the blue area. The red dotted line is the target function.

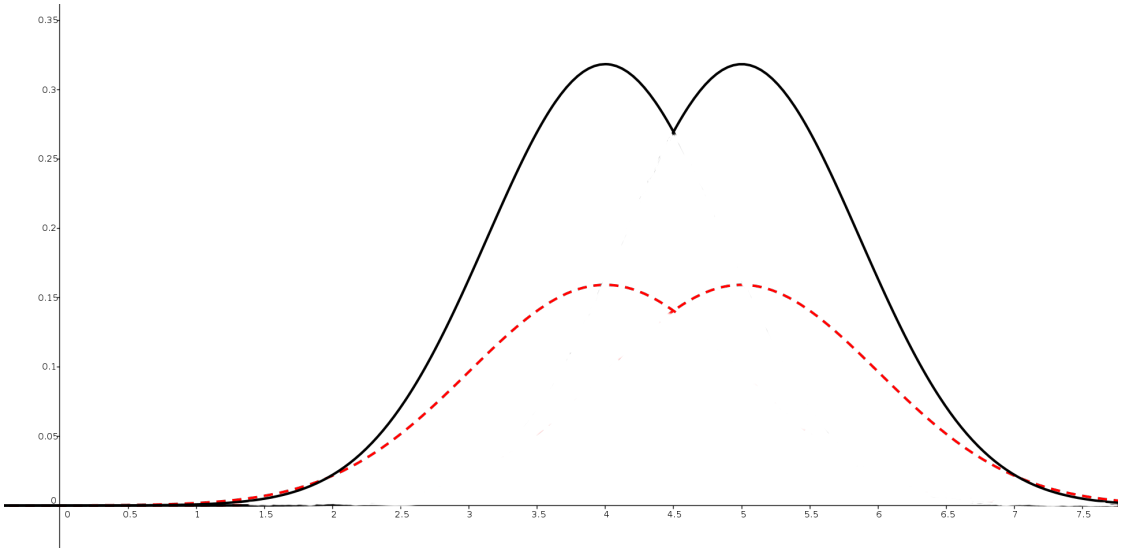


Figure 5.2: The second distribution having a worse average growth then the first one but always the right direction of growth, thus, the same amount of bell-shaped curves. The red dotted line is the target function.

Optimizations	NoG, first FM	NoG, second FM	NoG, third FM	NoG, fourth FM
Nothing	2000	8000	1600	7500
1	1700	7300	1300	7000
2	1800	7500	1400	6800
3	1800	7600	not used	not used
1,2	1500	7400	1000	6700
1,3	1400	69500	not used	not used
2,3	1600	7300	not used	not used
1,2,3	700	6000	not used	not used

Table 5.1: Table showing the speed up that can be achieved by the used optimizations. The number one stands for the first optimization strategy explained above and so on. The properties of the tested FMs can be seen in Section 5.1.1. *NoG* stands for "number of generations".

look like the target function, while Figure 5.4 would look alike. In order to prevent this situation, we put NFP values, which are in the same user defined range (called steps), into one category and count the number of values within this category. For example the user defines a range of 20, then all NFP values between 0 and 19 fall into one category. During our testing we discovered that the best range is between 10 and 20, depending on several different factors, such as number of variants, distribution of values, etc. Hence, we could not find any trend and give the user the possibility to adjust this value.

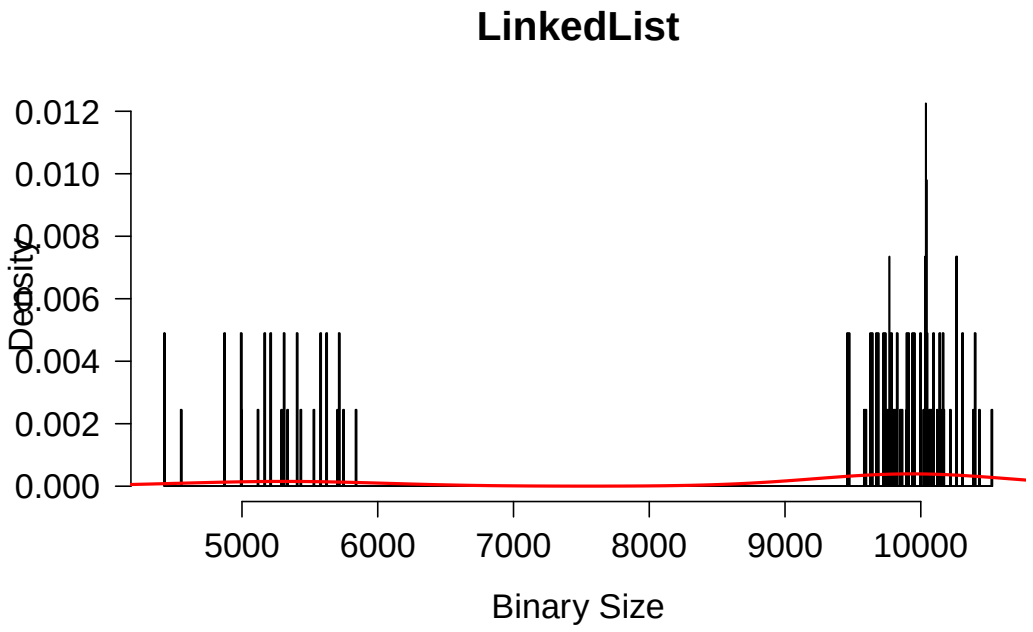


Figure 5.3: The distribution with a range (steps) of 1.

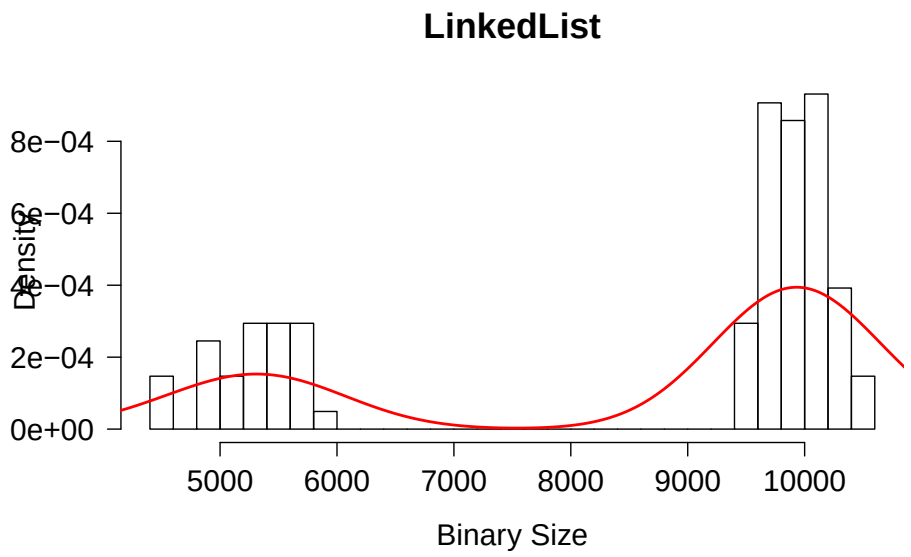


Figure 5.4: The distribution with a range (steps) of 20. For the sake of clarity of the bars the range in the graph is higher but it was calculated using a range of 20.

Quantity of Population

The quantity of the population, which stays the same during all generations and is very common among evolutionary algorithms, also has influence in the speed and accuracy of the algorithm due to the fact that a bigger population requires a longer execution time but also has a higher possibility of containing a good chromosome. As the user is free to adjust this value to its preferences a good compromise between speed and accuracy is around 100 to 200 chromosomes. When having a FM with a lower number of variants a population near 200 is advisable and vice versa.

Accuracy of the Resulting Distribution

The accuracy of the resulting distribution is used by one of the exit conditions. This value is the distance between the average growth of the distribution and the average growth of the target function.
Our tests are showing, that an accuracy of 6 is leading to an acceptable result. However, when lowering this value, better results are produced. But it is important to notice, that during our testing process, we discovered that a bisection of this value can increase the number of generations by a factor of 2.
Concluding, depending on the personal preference of the accuracy, a value of the interval $I =]0, 6] \in \mathbb{R}$ can be chosen.

5.1.2 Input and Effects

In Chapter 4, we introduced the user adjustable properties concerning the feature model. This section focuses on the effects that different inputs have on NFP distributions of the feature models. Furthermore we give a recommendation on the best input for the desired distribution as not all different inputs can create all types of

distributions.
The reason for that is fact that it is impossible to create a distribution with $n \in \mathbb{N}$ bell-shaped curves using a FM with $v < n$ variants ($v \in \mathbb{N}$). Its distribution can only have v peaks, meaning v bell-shaped curves.

Effects of Probabilities

As mentioned in Chapter 4 the four probabilities of a feature, being mandatory, optional, in an alternative relation or in an or relation, are completely adjustable by the user. This provides the possibility of generating all kinds of different initial distributions. As the sum of these four probabilities has to be exactly 100 there are four different basic ways of distributing the probabilities. Table 5.2 shows these allocations and the resulting NFP distributions.
These basic ways of distributing the probabilities are first of all to weight just one of them. This is shown in the table at number 1 to 4. As can be noticed, the less variants a FM has, caused by a higher number of variant lowering properties such as mandatory features, the higher the possibility of the distribution to be shaped like a uniform distribution.
Weighting two properties more than the others the results are quite the same as a higher number of mandatory features and features that exclude each other lowers the number of variants and provides the same distributions (to be seen at numbers 5 to 10).
Weighting three properties the strong variant lowering effects are smoothed so the resulting distributions are more likely to be shaped as a bell-shaped curve. The same is also true for weighting all properties the same.
Concluding the results of these measurements are on the one hand that a very low number of variants (e.g. 8 or 20) result in a distribution shaping an equipartition whereas a high number of variant result in a distribution similar to one, two or more bell-shaped curves. On the other hand if the user desires a specific distribution type (e.g. an equipartition) then a FM creating a very similar initial distribution (e.g. a FM containing many mandatory features) is advisable. In individual cases there might be another type of distribution as expected but these are exceptional cases.

Interactions

The feature generator also generates feature interactions, hence, we tested their influences on NFP distributions. As this is not the main part of the evaluation, we just focused on the potential of interactions to change an NFP distribution and do not give indepth explanation.
In Chapter 3 we discovered during our analysis of real-world Datasets that about 50 percent of all feature interactions had no influence on the NFP distribution due to their NFP value of 0. The remaining 50 percent split into the following intervals $I_{x,y} =]x \cdot z, y \cdot z] \in \mathbb{N}$ with $x, y \in [0, 1] \in \mathbb{R}$ and z being the highest NFP value in the FM: 15 percent of the NFP values of interactions v are $v \in I_{0,0.3}$, 25 percent are $v \in I_{0.3,0.7}$ and 10 percent $v \in I_{0.7,1}$. (Section 3.2.1 in Chapter 3)
Based on this data we tested the influence of interactions on the NFP distribution by taking the same FM with the same NFP values for the features and generated different numbers and types of interactions onto this FM. It should be noted that

Number	P. Alternative Relation	P. Or Relation	P. Optional	P. Mandatory	Type of Distribution
1	10	10	**70**	10	One and Two bell-shped curves
2	10	10	10	**70**	Equipartition
3	**70**	10	10	10	Equipartition, One bell-shaped curve
4	10	**70**	10	10	One and Two bell-shaped curves
5	**40**	**40**	10	10	One and Two bell-shaped curves
6	**40**	10	**40**	10	One and Two bell-shaped curves
7	**40**	10	10	**40**	Equipartition, One bell-shaped curve
8	10	**40**	**40**	10	One and Two bell-shaped curves
9	10	**40**	10	**40**	Equipartition, One and Two bell-shaped curves
10	10	10	**40**	**40**	One and Two bell-shaped curves
11	**30**	**30**	**30**	10	One and Two bell-shaped curves
12	**30**	**30**	10	**30**	One and Two bell-shaped curves
13	**30**	10	**30**	**30**	One and Two bell-shaped curves
14	10	**30**	**30**	**30**	One, Two and Three bell-shaped curves
15	**25**	**25**	**25**	**25**	One bell-shaped curve

Table 5.2: The table showing the resulting distribution respective to the input. The letter ”P” stands for probability. The more weighted properties are highlighted. The values are given in percent. The number of features was 20, and the percentage of CTCs and Interactions were kept low at 0%. The listed distributions are not the only ones that are generated by the given data but are most likely to be shaped like the ones mentioned. This does not mean that it is impossible for a FM with a high number of mandatory features, for example, to have a distribution shaped as three bell-shaped curve but it is very unlikely to happen, assuming that the NFP values are equal distributed. Hence, we did not mention them.

the distribution of the feature interactions is following: 75% of the interactions are of first order, 20% are of second order and the remaining 5% are of third order. We chose this distribution due to the fact that this is the most common one [SBL]. Figure 5.5 shows the NFP distribution of a FM without any feature interactions. The above mentioned distribution of NFP values concerning the feature interactions leads to the guess that those have a small influence on the behavior of the NFP distribution. However, when about 25% of all features are part of one or more interactions this behavior can dramatically change as seen in Figure 5.6. Listing 5.1 shows the interactions belonging to Figure 5.6. It can be noticed that there are only two interactions whose values are not 0 so those have the consequences that the NFP distribution changes. When increasing the number of features, interacting with each other, the influence of these interactions is increasing, for obvious reasons. Figure 5.7 shows the FM with 75% of its features interacting with each other. Concluding this analysis shows that a higher number of interactions is leading to a higher distraction of the distribution compared to the original one. This also explains the high number of NFP distributions shaped like three or more bell-shaped curves due to their interactions. Hence, we advise the users to include feature interactions into the generated FM when a NFP distribution consisting of several bell-shaped curves is desired.

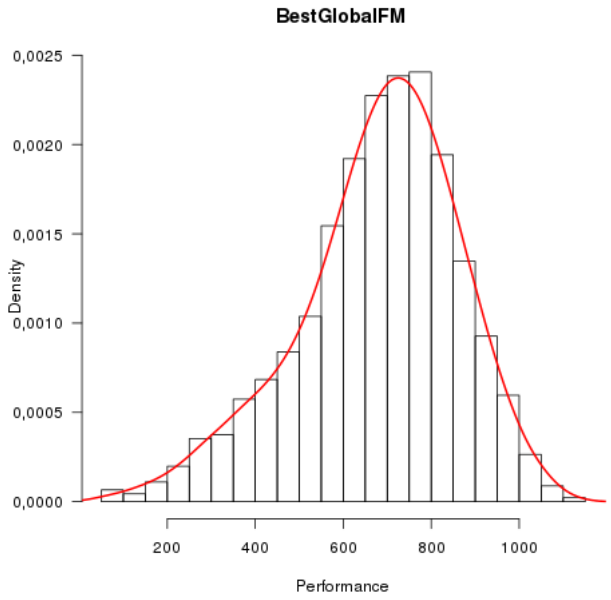


Figure 5.5: The NFP distribution of a Feature Model without any interactions. The FM consists of 20 features, 10% CTCs and the probability of a feature being optional is 70 %

Cross Tree Constraints

Cross Tree Constraints (CTC) limit the variability of a FM, when they are part of one. During our testing process we discovered, that a higher amount of CTCs is lowering the numbers of variants of the FM, thus, the probability of the FM, having a NFP distribution that is shaped like an equal distributed function, is increasing.

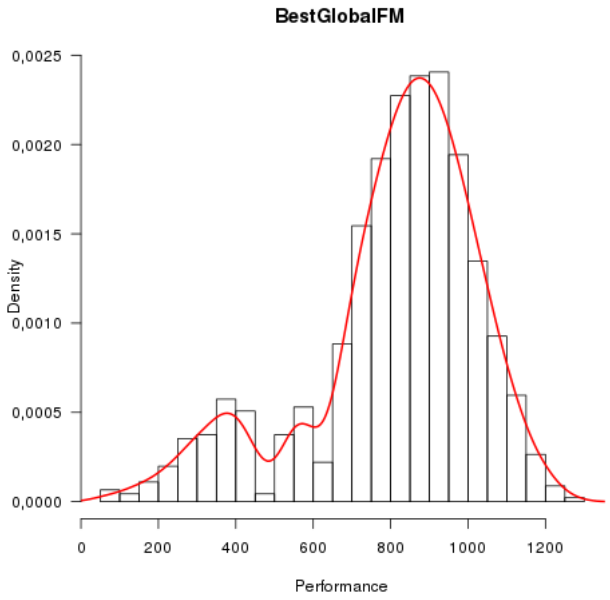


Figure 5.6: The NFP distribution of a Feature Model with 25% of its features being in an interaction. The FM consists of 20 features, 10% CTCs and the probability of a feature being optional is 70%

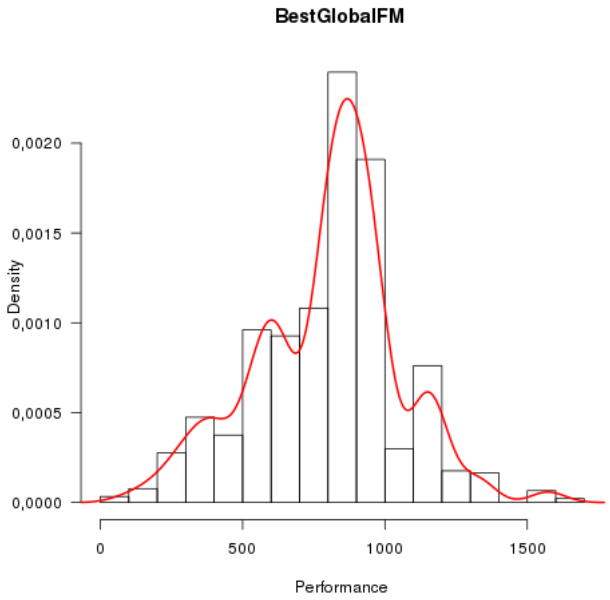


Figure 5.7: The NFP distribution of a Feature Model with 75% of its features being in an interaction. The FM consists of 20 features, 10% CTCs and the probability of a feature being optional is 70 %

Listing 5.1: Interactions belonging to Figure 5.6

```
<interacts feature0="F3" feature1="F7" name="I-21" value="87"/>
<interacts feature0="F3" feature1="F6" name="I-22" value="0"/>
<interacts feature0="F3" feature1="F8" name="I-23" value="63"/>
<interacts feature0="F3" feature1="F2" name="I-24" value="0"/>
<interacts feature0="F6" feature1="F3" feature2="F17" name="I-25"
        value="0"/>
```

Additionally, we tested the effects of a high CTC value (40% and more of the features are part of a CTC) on the resulting NFP distribution, with the FM properties of Table 5.2.
Our testing shows, that a percentage of 40% or higher of CTCs is increasing the probability of equal distributions by a high value, as most of the properties, that would result in a one or two bell-shaped curve, resulted in most cases in an equal distribution.
Concluding, if the desired output of the algorithm is a NFP distribution that is shaped like an equal distributed function, but for example there have to be still 70% of the features optional, then increasing the percentage of CTCs is an option.

5.2 Evaluation of SAT Sampling

We evaluated the SAT sampling explained in Chapter 3 in order to ensure the correctness of the sampling distribution compared to the actual distribution. We used the following setup for the evaluation process:
We took a feature model with all of its NFP values randomly chosen but staying the same during the testing. The FM had no interactions as they do not influence the SAT sampling.
It is important to know that when SAT sampling is turned on, 5000 variants are picked. We chose this number due to two reasons. First, when using a higher value such as 10000 the execution time of the algorithm doubles, but at the same time the accuracy is not doubling but growing considerably slower. Second, when using a lower value such as 1000 the accuracy of the sampling is affected too much due to just having one eighth of the original number of variants.
As a conclusion 5000 variants is a good compromise between time and accuracy, but this is only a reference value we defined so the user is free to change this number. Figure 5.8 shows the big aberration between the sampled distribution and the actual distribution.

In order to test the aberration of the distributions, we first generated all possible variants of the FM, calculated their NFP values and then translated this information into an NFP distribution. Second we used the SAT sampling to choose 10000, 5000 and 1000 variants and generated the corresponding NFP distributions. Afterwards we compared those by their distance of average growth. This process was done for every FM we tested. The following figure (Figure 5.9) shows the aberration between the actual and the 5000 variant distributions of a FM. The aberration stayed very similar for all tested FMs.

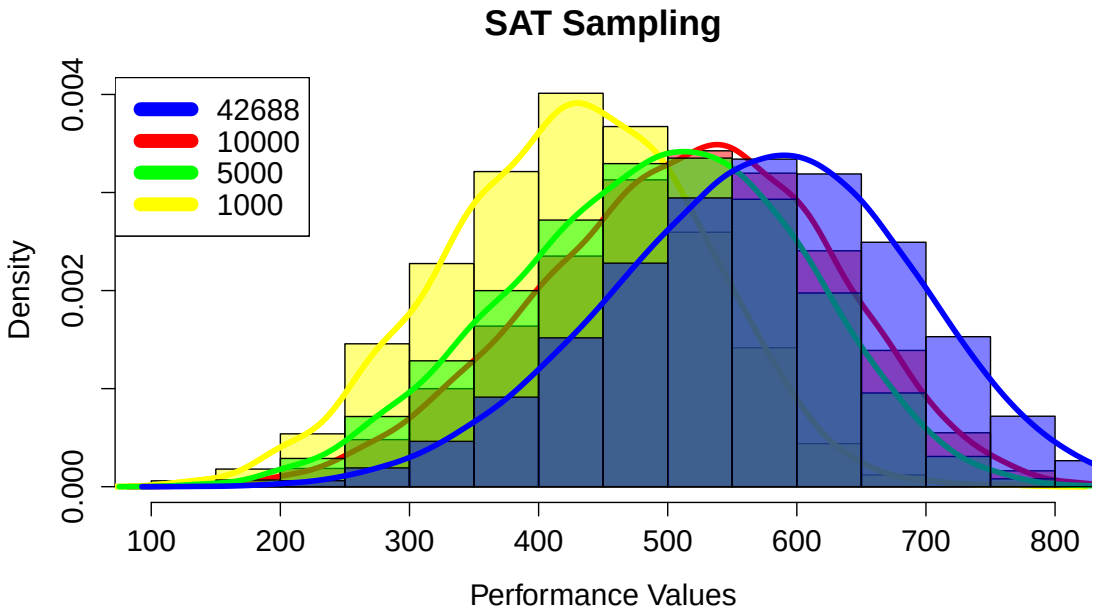


Figure 5.8: The NFP distributions of a Feature Model consisting of 20 features. The blue line represents the distribution among all variants, the red line the one among 5000 variants, the green one line among 5000 and the yellow one among 1000.

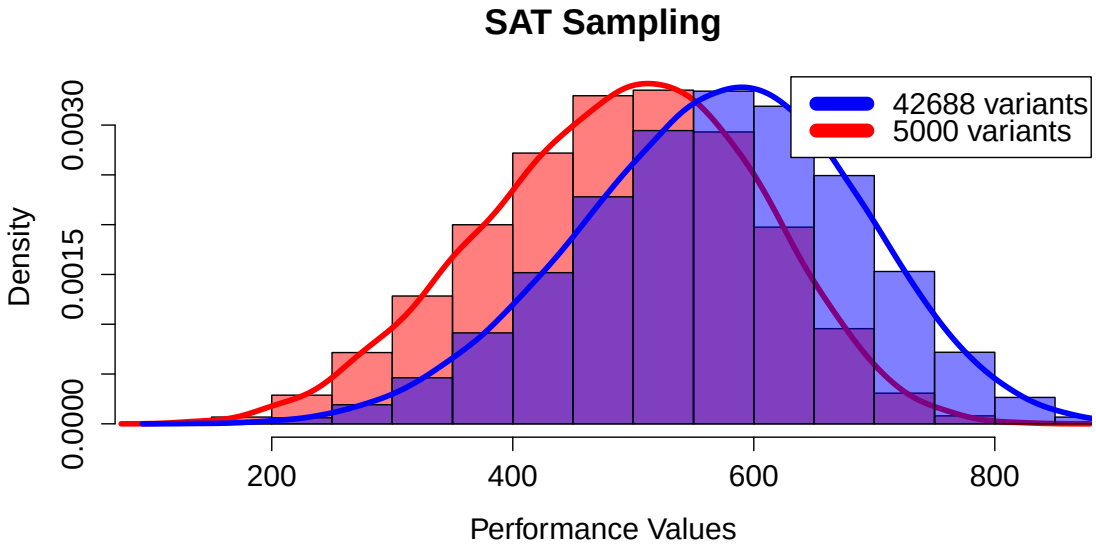


Figure 5.9: The NFP distributions of a Feature Model consisting of 20 features. The blue line represents the distribution among all variants, the red line the one among 5000 variants.

The difference between the distribution containing all variants and the one containing only 5000 is noticeable in the slight shifting of the whole function to the left. The actual shape of a one bell-shaped curve stayed the same recognizable by the similar height and width. Even though the FM with the sampled variants only consists of one eighth of the actual number of variants, the average aberration is 2.3 in this example. The other tested FMs also had an aberration of about 2.

6. Related Work

The subject of this thesis is relatively unexplored, what laid the basis for the motivation. There are, however, some papers to be considered as related work.
Soltani et al. generated an NFP distribution for their testing process by assigning NFP values to features that were produced by a random function with normal distribution. Depending on the number of variants and other factors, this distribution could result in an NFP distribution shaping, e.g. a bell-shaped curve. As this method, however, does not always produce real-world NFP distributions, e.g. it also could produce a distribution shaped as a linear growing function, our approach differs in that it always provides a user selectable real-world-based distribution [Sol12].
Segura et al. described the functionality of their testing and benchmarking tool called *BeTTy* and also emphasized the necessity of a testing tool for automated analysis of feature models. Their tool is a test suite, which is generating a user adjustable feature model, containing NFP values, and the corresponding correct properties of the FM (e.g. number of possible variants). Afterwards testing of analysis methods is done by giving the method the generated FM and comparing its output with the correct output created by BeTTy. Their approach of generating NFP values is very similar to the one Soltani et al. are using. They assign NFP values to features by a random function with pre-defined distributions. This differs to our approach in that we create real-world-based NFP values. Additionally, we add feature interactions to our generated FM. Hence, we create an even more realistic feature model generator [SGB+12].
Segura et al. also created a test suite called *FaMa*, which provides a set of test-cases in order to test analysis tools. Those test-cases are implementation independent, thus, providing a useful test suite. As this is the predecessor of BeTTy, however, they neither did include NFP values nor feature interactions, which separates this work from our approach. [SBRC10]
Segura et al. also had another approach of automatically generating test data for analysis tool by using a existing FM as a basis. This works by using a set of metaphoric relations and a test data generator. The generator then, is capable of generating neighbor FMs of the given FM, when knowing its set of products. Unfortunately, they also do not include NFP values or feature interactions. [SHBRC10]

Concluding the subject of testing with real-world NFP distributions is not jet fully discovered, thus, leaving a big gap and a huge need for further research. With our work we fill this gap, so further work will be inspired by our approach.

7. Conclusion

In this thesis we developed a test suite for the creation of a user adjustable FM with a user selectable real-world based NFP distribution.
The subject of this thesis stems from the need of a test suite for generating user adjustable feature models with real-world based NFP distributions. In order to proof this necessity, we analyzed the testing process of methods (which is described in literature) that analyze on feature models. As a result, we discovered that a vast majority of the methods are tested with random NFP distributions, thus, making it impossible to have confidence in the analysis results of those methods. Only one paper is using a different approach. Unfortunately, all papers analyzed, also do not include feature interactions in their FMs.
Additionally to the analysis of literature, we also analyzed the different types of NFP distributions of real-world datasets, in order to have reference functions our test suite can orientate on. Furthermore, we examined the typical behavior of NFP values of feature interactions, which differs from the one of features. All this data provided a basis for the creation and optimization of the resulting test suite.
This suite is a tool, which fills the wide open gap of testing tools and offers an easy and comfortable way of generating FMs, containing feature interactions, with real-world based NFP distributions. It offers a wide range of FMs due to the many user adjustable settings. The heart of this tool, an evolutionary algorithm, then transforms those settings into a FM containing the desired NFP distribution. Additionally one of the benefits of an evolutionary algorithm is the room for further improvements, hence, there is a huge potential for a later enhancement of user experience.
Afterwards, we evaluated our work, in particular the optimization strategies and the SAT sampling and additionally examined the effects of feature interactions and different types of FMs (e.g. FM with 20% optional features vs. FM with 70% optional features) on an NFP distribution.
With this test suite, testing can be easily done using real-world NFP distributions in order to ensure that the analysis tool, using this suite, is armed for all real-world situations.

8. Future Work

In this chapter, we focus on the work that could be done in the future in order to improve the algorithm or other components.

8.1 Fitness Function

The fitness function is one essential and critical part of an evolutionary algorithm, hence, there is a huge potential for improvement in all of its components, that can not be exploited to the full in this thesis. Reasons for that are time limitations, complexity and need for further research.

8.1.1 Initial Population

The initial population, defined in Chapter 4, has the task to randomly generate NFP values in order to assign those to the features and interactions contained in the before generated FM. This corresponds to the usual implementation of an evolutionary algorithm. However, there is another approach of generating an initial population which distributes NFP values very concerted in order to generate an initial population containing chromosomes that already have a very high similarity with the desired output. This reduces the execution time. [SHF+94]
Implementing this approach, we first would have to examine how the NFP values of the features have to be distributed when a certain NFP distribution of the variants is desired. That means, we would have to identify the typical NFP distribution of the features that leads to a certain NFP distribution of the variants. Thereby not only the NFP distribution of the features but also the number and composition of all possible variants have to be taken into account as they also influence the distribution of the variants.

8.1.2 Generating Functions

In Chapter 5, we explain the different metrics and their assets and drawbacks in matters of the comparison between the generated distribution and the target function. It also emphasizes the fact that we are limited in those metrics due to the circumstance that the generated distribution is described by a discrete key-value pair.
When we could generate a function out of this key-value pair, however, the comparison between the two functions would be much more accurate due to the possibility of using an integral or other metrics to compare those. This would boost the accuracy of the fitness function, thus, providing the basis for even better optimization techniques.
We know that this is already possible but only in a limited matter.

8.2 Optimization through Parallelization

The execution time is always a critical factor due to having very large populations, chromosomes with many genes, which is a high constant factor when comparing chromosomes, etc. Thus, it is advisable to parallelize some of the execution steps, as evolutionary algorithms are easily parallelized [Mit97].
When parallelizing, the following concepts can be used to improve execution time. The idea of this concept is to split the chromosomes into groups called *demes* and assign each core of a processor a deme. For all chromosomes the fitness is calculated regularly but a crossover between chromosomes of one deme are more likely to happen than between chromosomes of different demes [Tan89], [CHMR87]. As a benefit not only one local optimum would dominate the next generations but a wider variety of chromosomes would arise and might lead to a bigger variety of even better chromosomes in the next generations.

8.3 Mutation and Crossover

Mutation and crossover are the instruments of an evolutionary algorithm to generate the chromosomes of the next generation with a higher fitness then their predecessors. Hence, there are several different approaches of those operations. The variants used in this thesis are called ”single-point crossover” and ”point mutation” and explained in Chapter 4.
In a future work different crossover and mutation types like ”uniform crossover”, where not only one set of genes is exchanged but several ones, of different length, can be tested [Mit97]. Also normal distributed mutation, where the gene being mutated is selected by a random number which is normal distributed, is an option [SHF+94].
Additionally other optimization instruments, such as operators using domain specific rules, can be introduced [Jan93]. Those have the ability of changing single genes not randomly but tailored to the target function, in order to generate a chromosome with a higher fitness.

8.4 Extending the Feature Model

Feature Models can have several different components, such as features with NFP values and CTC. In future work the components we use in this thesis can be extended. In particular numeric features and NFP values which are composed of the NFP values of other features. All those improvements would lead to a generation of even more realistic feature models, thus, improving the quality of the test suite.

A. Appendix

Visual Representation of the NFP distributions

This is an additionally section to inform the reader about the problem, we had with the visual representation of the NFP distributions. In order to visualize those, we used the statistical program called *R*. When generating the histogram of the below seen NFP distributions, the only way of displaying this red density curve, was to change the representation of the y-axis. This went from displaying the frequency of the values of the x-axis, to displaying the density of those. This means, that the sum of the area of all bars adds up to 1 and therefore, the values of the y-axis are not as expected. Until the deadline of this thesis, we unfortunately did not find any solution to that problem.
This problem is, however, not as problematic, due to the fact that the course of the function is visible, which is the main purpose of those histograms in the subject of this thesis.

Implementation Dependent Constraints

When starting the graphical user interface (GUI) of this test suite, the user will be prompted to enter some property values for the FM. Hereby, it is important to notice, that the sum of the probabilities of a feature being optional, being mandatory, are part in an or and alternate relation have to add up to exactly 100%.
the same is true for changing the percentages of interactions of different order. Those three values also have to sum up to 100%. Additionally, when the user wants to change those values the check mark above has to be checked.
When the user enters the settings of the algorithm, and is prompted to enter the target function, there are some constraints to be noticed:

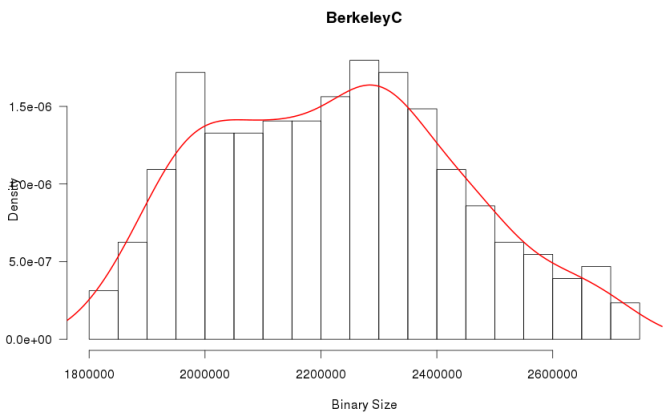
- The variable of the function has to be named x. For example $2 * x$. It must not be another name like ”y”.

- When calculating the maximum of two or more functions, then the word ”max” has to be used and the functions have to be surrounded by brackets and separated by commas, like ”max(a,b,c)”. The same goes for the minimum, as the word is ”min”.

When including all the extern libraries, in order to run the program, the user needs to have the program *R* installed and furthermore the library called *JRI* has to be installed. The installation process is depending on the operating system, thus, we advice the user to use the provided installation guide [1].
As an important notice: The file called ”FMInteractionEinlesen.R” has to be in the same path, the program is executed from.

A.1 All analyzed real-world NFP Distributions

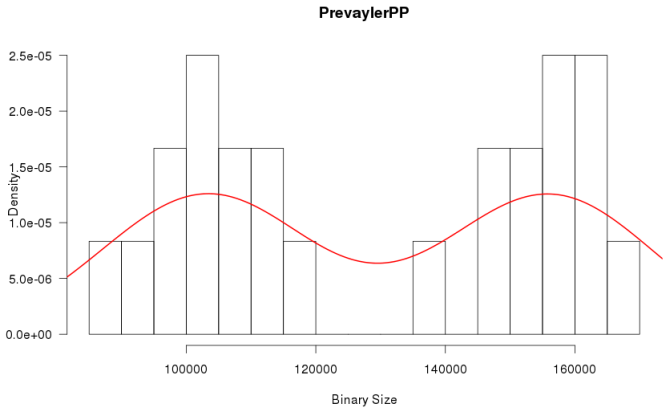
In this section, we list all the analyzed real-world NFP distributions together with some additional information.
Due to a Latex bug, the remaining NFP distribution could not be loaded. However, all distributions are also contained in a digital form on a CD.
Due to bugs in SPLConqueror, some variants were not calculated and some data was not accepted at all.



BerkeleyC - NFP: Binary Size

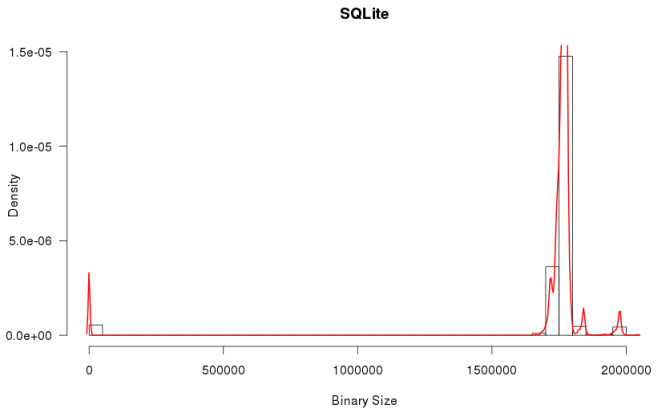
- Number of Features 18
- Number of Mandatory Features 2
- Number of Optional Features 7
- Number of Features in Or Relations 0
- Number of Features in Alternative Relations 9
- Number of Variants 2560

[1] http://rforge.net/JRI/



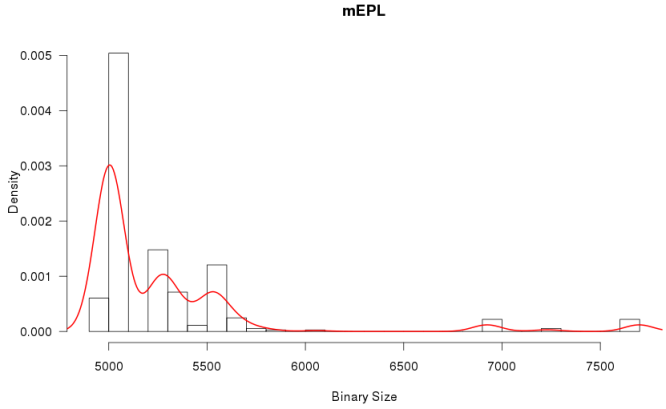
PrevaylerPP - NFP: Binary Size

- Number of Features 6
- Number of Mandatory Features 1
- Number of Optional Features 5
- Number of Features in Or Relations 0
- Number of Features in Alternative Relations 0
- Number of Variants 24



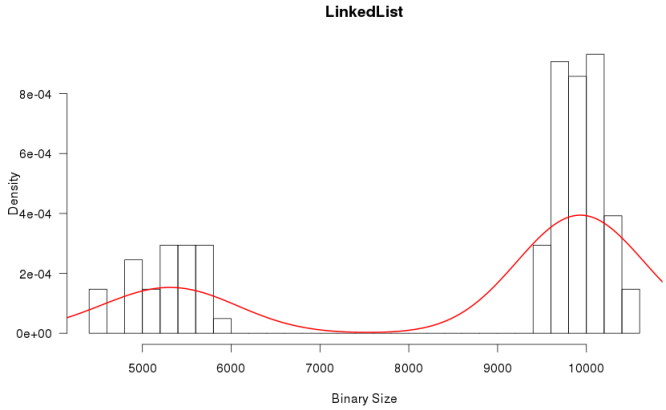
SQLite - NFP: Binary Size

- Number of Features 88
- Number of Mandatory Features 5
- Number of Optional Features 77
- Number of Features in Or Relations 0
- Number of Features in Alternative Relations 6
- Number of Variants not applicable



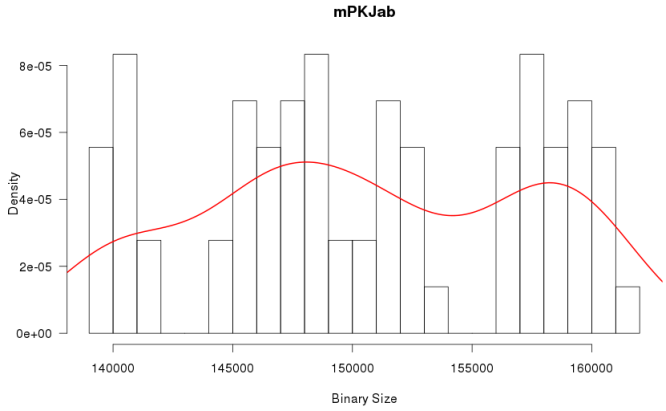
mEPL - NFP: Binary Size

- No Data found!



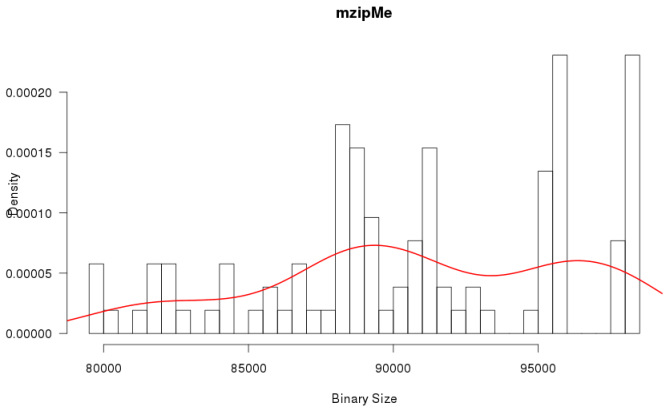
LinkedList - NFP: Binary Size

- No data found!



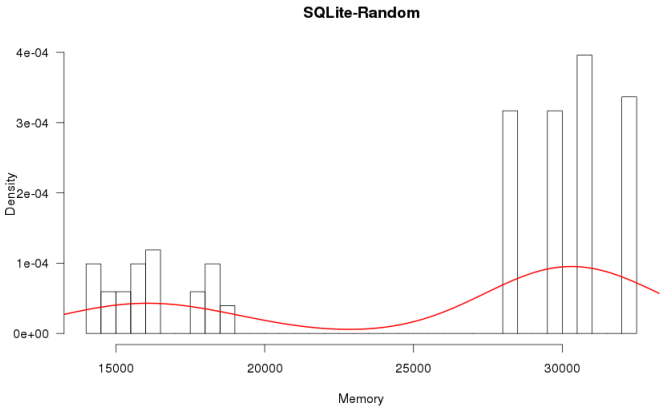
mPKJab - NFP: Binary Size

- Number of Features 1
- Number of Mandatory Features 4
- Number of Optional Features 7
- Number of Features in Or Relations 0
- Number of Features in Alternative Relations 0
- Number of Variants 72



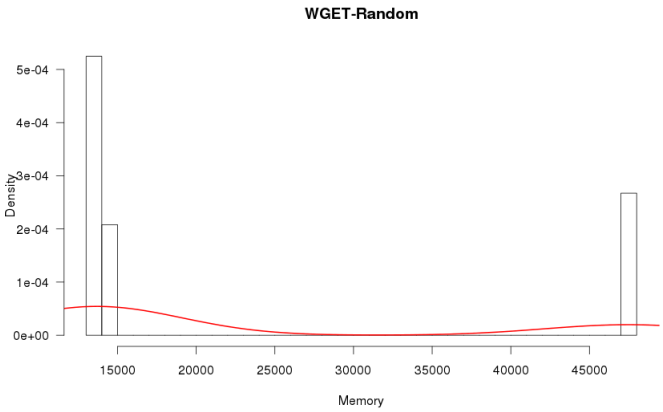
mZipMe - NFP: Binary Size

- Number of Features 8
- Number of Mandatory Features 1
- Number of Optional Features 7
- Number of Features in Or Relations 0
- Number of Features in Alternative Relations 0
- Number of Variants 128



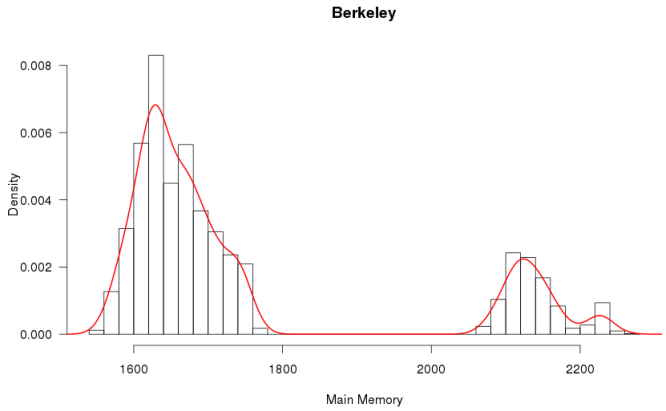
SQLite - NFP: Memory Consumption

- Number of Features 39
- Number of Mandatory Features 9
- Number of Optional Features 15
- Number of Features in Or Relations 0
- Number of Features in Alternative Relations 15
- Number of Variants not applicable



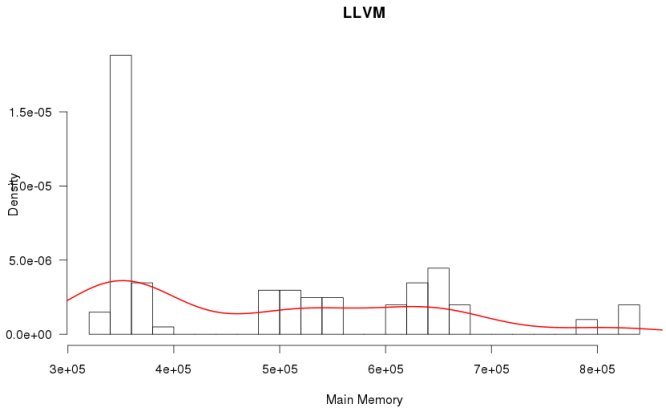
WGet - NFP: Memory Consumption

- Number of Features 16
- Number of Mandatory Features 2
- Number of Optional Features 11
- Number of Features in Or Relations 0
- Number of Features in Alternative Relations 3
- Number of Variants 5120



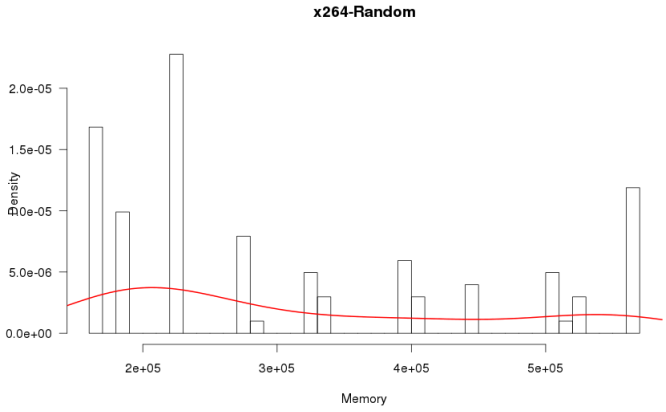
BerkeleyC - NFP: Memory Consumption

- Number of Features 18
- Number of Mandatory Features 2
- Number of Optional Features 7
- Number of Features in Or Relations 0
- Number of Features in Alternative Relations 9
- Number of Variants 2560



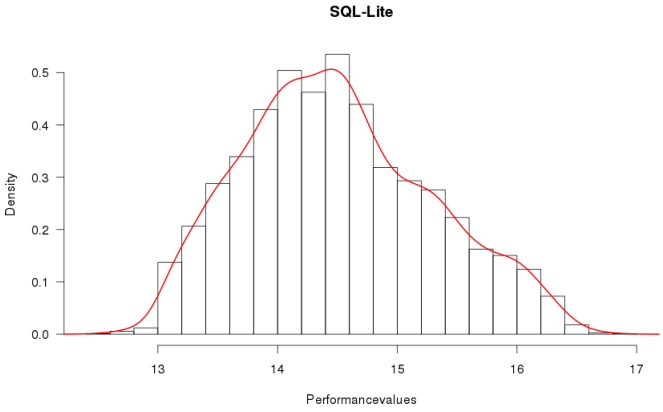
LLVM - NFP: Memory Consumption

- Number of Features 11
- Number of Mandatory Features 1
- Number of Optional Features 10
- Number of Features in Or Relations 0
- Number of Features in Alternative Relations 0
- Number of Variants 1024



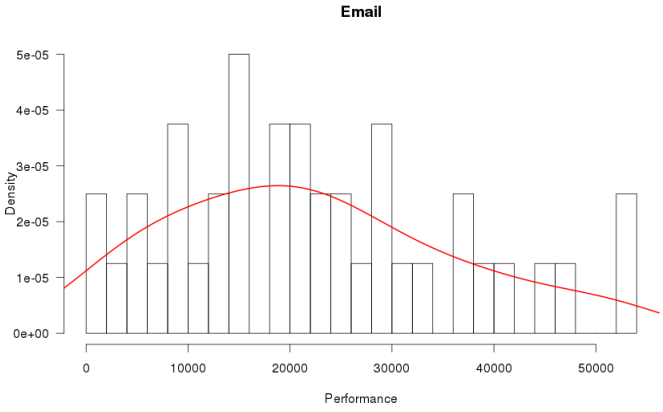
x264 - NFP: Memory Consumption

- No data found!



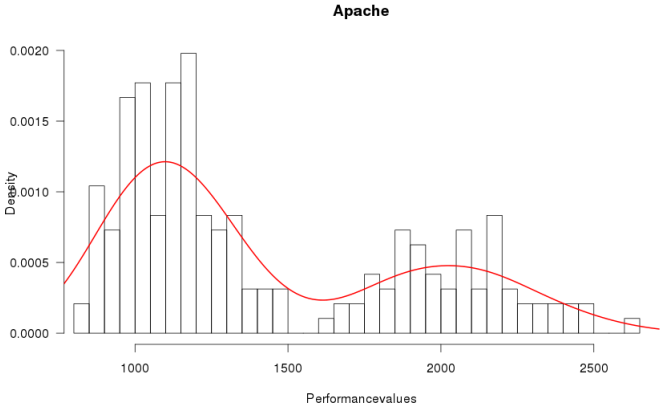
SQLite - NFP: Performance

- Number of Features 88
- Number of Mandatory Features 5
- Number of Optional Features 77
- Number of Features in Or Relations 0
- Number of Features in Alternative Relations 6
- Number of Variants not applicable



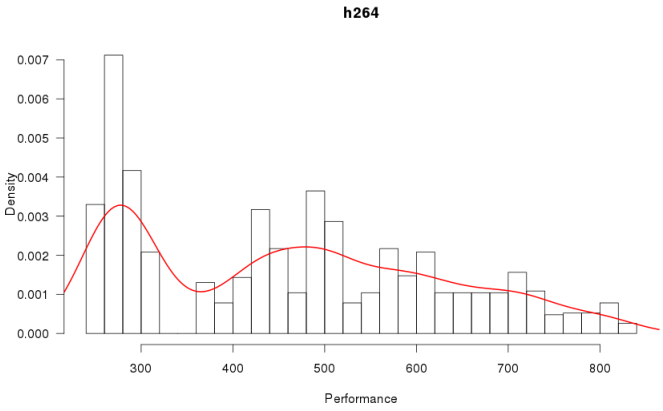
Email - NFP: Performance

- Number of Features 9
- Number of Mandatory Features 1
- Number of Optional Features 8
- Number of Features in Or Relations 0
- Number of Features in Alternative Relations 0
- Number of Variants 40



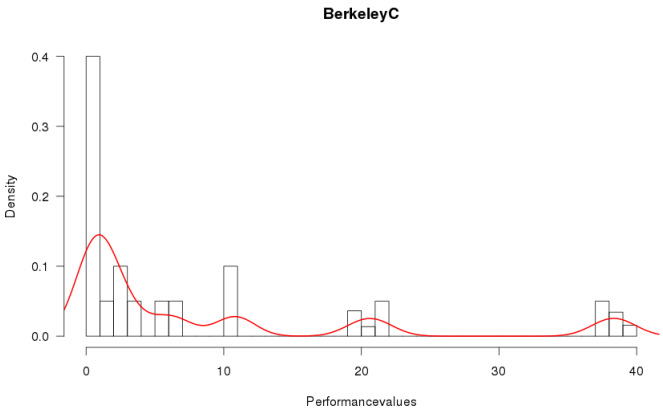
Apache - NFP: Performance

- Number of Features 9
- Number of Mandatory Features 1
- Number of Optional Features 8
- Number of Features in Or Relations 0
- Number of Features in Alternative Relations 0
- Number of Variants 192



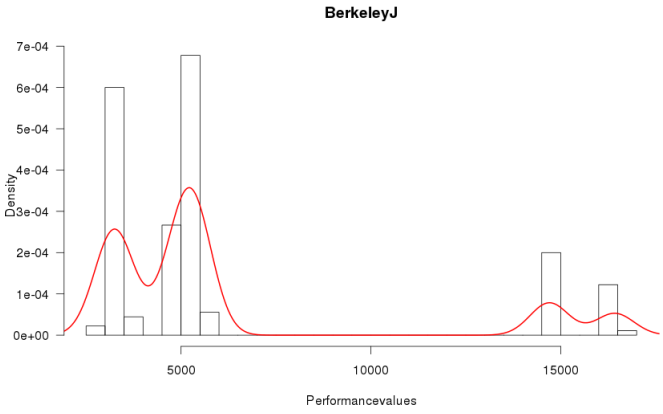
h264 - NFP: Performance

- Number of Features 16
- Number of Mandatory Features 3
- Number of Optional Features 7
- Number of Features in Or Relations 0
- Number of Features in Alternative Relations 6
- Number of Variants 1152



BerkeleyC - NFP: Performance

- Number of Features 18
- Number of Mandatory Features 2
- Number of Optional Features 7
- Number of Features in Or Relations 0
- Number of Features in Alternative Relations 9
- Number of Variants 2560



BerkeleyJ - NFP: Performance

- Number of Features 32
- Number of Mandatory Features 11
- Number of Optional Features 8
- Number of Features in Or Relations 0
- Number of Features in Alternative Relations 13
- Number of Variants 6480

Bibliography

[ABM+13] Michał Antkiewicz, Kacper Bąk, Alexandr Murashkin, Rafael Olaechea, Jia Hui (Jimmy) Liang, and Krzysztof Czarnecki. Clafer tools for product line engineering. *Proceedings of the 17th International Software Product Line Conference co-located workshops on - SPLC '13 Workshops*, page 130, 2013. (cited on Page 8)

[Ale03] Ian Alexander. Misuse cases help to elicit non-functional requirements. *Computing and Control . . .*, pages 1–10, 2003. (cited on Page 8)

[BMO09] Joerg Bartholdt, Marcel Medak, and Roy Oberhauser. Integrating Quality Modeling with Feature Modeling in Software Product Lines. *2009 Fourth International Conference on Software Engineering Advances*, pages 365–370, September 2009. (cited on Page 8)

[BTRC05a] David Benavides, Pablo Trinidad, and A Ruiz-Cortés. Automated reasoning on feature models. *Advanced Information Systems . . .*, 01:491–503, 2005. (cited on Page 8 and 9)

[BTRC05b] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortés. Automated reasoning on feature models. In *Advanced Information Systems Engineering*, pages 491–503. Springer, 2005. (cited on Page 1 and 2)

[CdPL09] Lawrence Chung and Julio Cesar Sampaio do Prado Leite. On non-functional requirements in software engineering. In *Conceptual modeling: Foundations and applications*, pages 363–379. Springer, 2009. (cited on Page 8)

[CE00] Krzysztof Czarnecki and Ulrich W Eisenecker. Generative programming. 2000. (cited on Page 5)

[CHMR87] James P Cohoon, Shailesh U Hegde, Worthy N Martin, and D Richards. Punctuated equilibria: a parallel genetic algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 148–154. L. Erlbaum Associates Inc., 1987. (cited on Page 46)

[CKMRM03] Muffy Calder, Mario Kolberg, Evan H Magill, and Stephan Reiff-Marganiec. Feature interaction: a critical review and considered forecast. *Computer Networks*, 41(1):115–141, 2003. (cited on Page 7)

[CPFD01] Denis Conan, Erik Putrycz, Nicolas Farcet, and Miguel DeMiguel. Integration of non-functional properties in containers. In *Proceedings of the Sixth International Workshop on Component-Oriented Programming (WCOP)*, volume 1, page 2, 2001. (cited on Page 8)

[EMB07] Leire Etxeberria, GS Mendieta, and Lorea Belategi. Modelling Variation in Quality Attributes. *VaMoS*, 2007. (cited on Page 8)

[GWW+11] Jianmei Guo, Jules White, Guangxin Wang, Jian Li, and Yinglin Wang. A genetic algorithm for optimized feature selection with resource constraints in software product lines. *Journal of Systems and Software*, 84(12):2208–2221, December 2011. (cited on Page 9)

[Jan93] Cezary Z Janikow. A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, 13(2-3):189–228, 1993. (cited on Page 46)

[KAK09] Christian Kästner, Sven Apel, and Martin Kuhlemann. A Model of Refactoring Physically and Virtually Separated Features. In *Proceedings of the International Conference on Generative Programming and Component Engineering (GPCE)*, pages 157–166. ACM, October 2009. (cited on Page 2)

[MARC13] Alexandr Murashkin, Michał Antkiewicz, Derek Rayside, and Krzysztof Czarnecki. Visualization and exploration of optimal variants in product line engineering. *Proceedings of the 17th International Software Product Line Conference on - SPLC '13*, page 111, 2013. (cited on Page 9)

[Mit97] Tom M Mitchell. Machine learning. wcb, 1997. (cited on Page 9 and 46)

[RFBRC+11] Fabricia Roos-Frantz, David Benavides, Antonio Ruiz-Cortés, André Heuer, and Kim Lauenroth. Quality-aware analysis in product line engineering with the orthogonal variability model. *Software Quality Journal*, 20(3-4):519–565, August 2011. (cited on Page 9)

[RFBRC+12] Fabricia Roos-Frantz, David Benavides, Antonio Ruiz-Cortés, André Heuer, and Kim Lauenroth. Quality-aware Analysis in Product Line Engineering with the Orthogonal Variability Model. *Software Quality Journal*, 20(3-4):519–565, 2012. (cited on Page 1)

[RNMM08] Mikko Raatikainen, E Niemelä, V Myllärniemi, and T Männistö. Svamp-An Integrated Approach for Modeling Functional and Quality Variability. *VaMoS*, 2008. (cited on Page 9)

[SBL] Gunter Saake, Don Batory, and Christian Lengauer. Measuring and predicting non-functional properties of customizable programs. (cited on Page 36)

[SBRC10] Sergio Segura, David Benavides, and Antonio Ruiz-Cortés. Fama test suite v1. 2. *ISA Research Group*, 2010. (cited on Page 41)

[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 U.W. Eisenecker, S. Apel, and S. Gnesi, editors, *Sixth International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'12)*, page 63–71, Leipzig, Germany, 2012. ACM, ACM. (cited on Page 20 and 41)

[SHBRC10] Sergio Segura, Robert M Hierons, David Benavides, and Antonio Ruiz-Cortés. Automated Test Data Generation on the Analyses of Feature Models: A Metamorphic Testing Approach. In *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on*, pages 35–44. IEEE, 2010. (cited on Page 41)

[SHF+94] Eberhard Schöneburg, Frank Heinzmann, Sven Feddersen, et al. Genetische algorithmen und evolutionsstrategien. *Bonn: Addison-Wesley*, 1994. (cited on Page 9, 45, and 46)

[SIMA13] Abdel Salam Sayyad, Joseph Ingram, Tim Menzies, and Hany Ammar. Scalable product line configuration: A straw to break the camel's back. *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 465–474, November 2013. (cited on Page 9)

[SKK+12] Norbert Siegmund, Sergiy S Kolesnikov, Christian Kästner, Sven Apel, Don Batory, Marko Rosenmüller, and Gunter Saake. Predicting performance via automated feature-interaction detection. In *Proceedings of the 34th International Conference on Software Engineering*, pages 167–177. IEEE, 2012. (cited on Page 11)

[Sol12] Samaneh Soltani. Towards automated feature model configuration with optimizing the non-functional requirements. 2012. (cited on Page 9 and 41)

[SRK+11] Norbert Siegmund, Marko Rosenmuller, Christian Kastner, Paolo G Giarrusso, Sven Apel, and Sergiy S Kolesnikov. Scalable prediction of non-functional properties in software product lines. In *Software Product Line Conference (SPLC), 2011 15th International*, pages 160–169. IEEE, 2011. (cited on Page 11)

[SRK+13] Norbert Siegmund, Marko Rosenmüller, Christian Kästner, Paolo G Giarrusso, Sven Apel, and Sergiy S Kolesnikov. Scalable prediction of non-functional properties in software product lines: Footprint and memory consumption. *Information and Software Technology*, 55(3):491–507, 2013. (cited on Page 11)

[SRP] D. Streitferdt, M. Riebisch, and K. Philippow. Details of formalized relations in feature models using OCL. *10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, 2003. Proceedings.*, pages 297–304. (cited on Page 9)

[SvRA13] Norbert Siegmund, Alexander von Rhein, and Sven Apel. Family-based performance measurement. In *Proceedings of the 12th international conference on Generative programming: concepts & experiences*, pages 95–104. ACM, 2013. (cited on Page 11)

[Tan89] Reiko Tanese. Distributed genetic algorithms. In *Proceedings of the third international conference on Genetic algorithms*, pages 434–439. Morgan Kaufmann Publishers Inc., 1989. (cited on Page 46)

[WDS09] Jules White, Brian Dougherty, and DC Schmidt. Selecting highly optimal architectural feature sets with filtered cartesian flattening. *Journal of Systems and Software*, (August 2008):1–36, 2009. (cited on Page 9)

[WSNW] Jules White, Douglas C Schmidt, Andrey Nechypurenko, and Egon Wuchner. Optimizing and Automating Product-Line Variant Selection for Mobile Devices. (cited on Page 9)

Eidesstattliche Erklärung:

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

Name

Passau, den 27. Juni 2014