Approaches to Multi-Objective Optimization of Formal Specifications

(Ansätze zur Multikriteriellen Optimierung Formaler Spezifikationen)

Autor:
Simon Struck
January 17, 2012

Masterarbeit

Betreuer:
Jun.-Prof. Dr. Frank Ortmeier

Gutachter:
Jun.-Prof. Dr. Frank Ortmeier
Prof. Dr.-Ing. Jana Dittmann
Struck, Simon:

Approaches to Multi-Objective Optimization of Formal Specifications
(Ansätze zur Multikriteriellen Optimierung Formaler Spezifikationen)

Masterarbeit, Otto-von-Guericke-Universität
Magdeburg, 2012.
## Contents

1. Introduction .................................................. 1
   1.1. Motivation .......................................... 1
   1.2. Formal Specification ................................ 1
   1.3. Optimization of a Specification .................... 2
   1.4. Problem Description .................................. 3

2. Multi-Objective Optimization with Genetic Algorithms .... 5
   2.1. Multi-Objective Optimization Problem ............... 5
   2.2. Multi-Objective Evolutionary Algorithm ............. 9
      2.2.1. Origin of Evolutionary Algorithms ............... 9
      2.2.2. Evolutionary Algorithms for Optimization ....... 10
      2.2.3. Encoding of the Chromosome .................... 11
      2.2.4. Classification of Evolutionary Algorithms ....... 11
   2.3. Crossover Operators ................................ 12
      2.3.1. Single Point Crossover ......................... 13
      2.3.2. Simulated Binary Crossover Operator .......... 13
   2.4. Non-Dominated Sorting Genetic Algorithm II ....... 15

3. Safety Optimization ........................................ 21
   3.1. Safety Analysis .................................... 21
   3.2. Safety Analysis Modeling Language ................. 23
      3.2.1. SAML Syntax .................................. 24
      3.2.2. SAML Semantics ................................ 26
      3.2.3. Analysis of a SAML Model ..................... 29
   3.3. Optimizable Safety Analysis Modeling Language .... 29
      3.3.1. Syntax ....................................... 29
      3.3.2. Semantics .................................... 31
List of Figures

2.1. The Pareto dominance ............................................. 7
2.2. The Pareto Optimal Set (left) and the Pareto Front (right) .... 8
2.3. The creation of an offspring generation from a given population . 10
2.4. One-point crossover of two binary strings ........................ 13
2.5. The value of the spread factor indicates the location of the children in relation to their parents ................................. 14
2.6. The probability distribution of the spread factor $\beta$ ............... 14
2.7. Distribution for two children generated from a pair of parents, with the values three and seven .................................... 15
2.8. General idea of non-dominated sorting ............................... 16
2.9. Calculation of the crowding distance ................................ 17
2.10. NAGA-II approach to create a next generation from the current population ................................................................. 19
2.11. Solutions that are dropped by the NSGA-II algorithm are preserved in the external storage ................................. 20
3.1. Block diagram of a reactive system that is interacting with the environment ................................................................. 21
3.2. A simplified version of the SAML syntax in extended Backus-Naur form ................................................................. 25
3.3. A simple SAML model .................................................. 26
3.4. A simple SAML model with parameters ............................. 30
3.5. Syntax of the optimizable safety analysis modeling language ....... 33
4.1. Block diagram of a simple artificial neuron with three inputs .... 36
4.2. The shape of the Sigmoid, Linear and Sign activation functions ... 37
4.3. A multilayer feed forward artificial neural network with two hidden layers ........................................... 38
4.4. The classification according to the and and the exclusive or function 39
4.5. Simple network with two bias neurons, depicted as small solid black circles ........................................... 40
4.6. Explanation of the symbols used in the derivation of the backpropagation algorithm ........................................... 42
4.7. Estimation strategy according to Deb et al. ................................. 47
4.8. The estimator is used as a filter prior to the expensive evaluation function ........................................... 48
5.1. Block diagram of the uninterruptible power supply. .................. 51
5.2. The Pareto front of the USV model with logarithmic scaling on both axis. ........................................... 54
5.3. Both objectives of the USV plotted over the number of batteries... 55
5.4. Hazard probability of the USV plotted over the diesel start threshold 55
5.5. System costs of the USV plotted over the diesel start threshold ... 56
5.6. Both objectives of the USV plotted over the battery failure probability 56
5.7. Hazard probability of the USV plotted over the generator type ... 57
5.8. System costs of the USV plotted over the generator type ... 57
6.1. The connection between the optimization related and SAML related code ........................................... 59
6.2. Class diagram of the SAML related code ................................. 60
6.3. Class diagram of the optimizer related code ............................... 61
6.4. Class diagram of the estimator related code ............................... 62
6.5. The numerical data types used by the optimizer ........................... 63
7.1. Worse case interpolation of a Pareto front. .............................. 65
7.2. Distance between a point and a Pareto front that is given as a set of points. ........................................... 67
7.3. Graphical representation of the results of the convergence analysis . 68
7.4. Graphical representation of the convergence analysis with error bars 69
7.5. Graphical representation of Table 7.3 ................................. 71
List of Tables

7.1. Configuration of the convergence speed experiment ........ 68
7.2. Configuration of the experiments with variable population size. . . 70
7.3. Comparison of different population-size settings with and without estimator. ................................................................. 70
1. Introduction

1.1. Motivation

A specification is an artifact, which describes the expected behavior of a system. In engineering, a specification can be expressed, among others, in terms of a human readable text, block diagrams, schematics or process diagrams. Even combinations of different styles are possible.

Problems usually appear when the specification is ambiguous. This is especially a problem with human readable texts, as it may be interpreted differently by readers from different domains. To compensate different interpretations of a specification, prototypes are built and tests are performed. However, prototypes and tests are not always feasible. A test could cause high costs or imply unacceptable safety critical aspects.

An example which is easy to understand is the computer system on board of an airplane. A real test can only be performed if the plane is flying. If the system fails, at least the pilot is in great danger. As a result, there must be some quality insurance before the first test run.

Assuming the specification of an engineer is correct, verification can be used to ensure the correct functionality of a system with respect to the specification. However, to argue about the functionality of a system, a specification with a distinct semantic is required. This is where formal specification comes in. The specification can then be expressed in a notation with a mathematically defined and well-founded semantic. As a result, the specification is no longer ambiguous.

1.2. Formal Specification

A formal specification can be expressed in terms of a finite state automaton. Such a notation defines all possible states of a system and all allowed transitions between the states. An example for a formal specification language is the safety analysis modeling language (SAML)[1].

Another formal specification language is LUSTRE [2]. In contrast to SAML, LUSTRE specifies the behavior of a system based on the data flow and operations performed on the data.

There still is a critical point, regarding the correctness of the specification. It might be feasible to state a finite state machine based specification for a simple system with only a few states. However, modern computer-controlled systems easily reach a very large state space. Thus, an engineer is not able to determine the
correctness of the specification. It is likely that certain critical states are overseen, or the system may run into situations that were not encountered by the engineers. However it is often possible to state simple properties that must hold for a certain specification. These properties are expected to be fulfilled under all circumstances. Now the specification itself can be verified against the properties. This helps to ensure the correctness of the specification.

In case of the airplane computer system a safety property could be that the thrust reversal must not be activated as long as the airplane is off the ground. In automation technology a safety property could say that the machine must be turned off, as long as the safety housing is opened. Such properties can be expressed in terms of temporal logic. The analysis of the specification with respect to the property often leads to the question whether the system can reach a critical state (a state were the safety properties are violated) or not.

If the system is verified against the specification and the specification is verified against the properties, it is ensured that the system also fulfills the properties. In this chain of verification the specification is also often referred to as a model. In some cases the model can even be transformed into an implementation.

A famous model based programming and verification tool is Esterel SCADE\(^1\). SCADE relies on the LUSTRE language. The SCADE-Suite enables the engineer to enter a model (specification) and verify this model against certain properties. When the model fulfills all properties the SCADE-Suite can automatically transform the specification into a c-program. The correctness of the transformation algorithm has been proved, so that the resulting c-code has the same semantic as the (LUSTRE) model.

This far only qualitative aspects, such as if a specification fulfills certain properties, are considered. This analysis covers typical design failures of a system. However, it is a matter of fact that every technical system can fail. At least if a part of the system breaks down. Quantitative analysis techniques are used to cover probabilistic behavior. Quantitative analysis techniques enable the analysis of the influence of certain transition probabilities on the probability that a critical state is reached. This enables the determination of the total failure probability of the system based on the known failure probabilities of all components.

Further details on qualitative and quantitative analysis (as far as they are required for the understanding of this thesis) are explained in Section 3.

1.3. Optimization of a Specification

Up to this point the model or the specification is treated as an invariant artifact. This means the specification usually represents the idea of the engineer who is designing the system. However, common questions asked during the design time of a system could be (among others):

\(^1\)http://www.esterel-technologies.com/products/scade-suite/
• Is it possible to increase the system safety?
• Can the overall costs be reduced?
• How can the performance be increased?
• Is it safer and possibly cheaper, to use several redundant but cheap sensors which might be less reliable instead of one single reliable but expensive sensor?
• In case several redundant sensors are used, what is the best compromise between the number of sensors (safety/reliability) and the overall system costs?
• Would a decrease of the velocity (or increase of the safety margins) lead to the same safety level but costs less?

Such questions are, at least for complex systems, often not easy to answer. An approach to find solutions to these problems is to use a parameterized model (or specification). From one point of view, this leads to specifications of product lines or families of systems. In such a family all variants share a basic part of the specification and each variant has certain specialties. The main idea in this thesis is the opportunity to optimize the specification along certain objectives.

Possible optimization objectives could be safety and performance maximization, or cost minimization. It is quite clear that certain parameters of a system influence its safety, performance and/or costs. Prominent examples of such parameters are maximum allowed velocities, safety margins, sensor accuracy or component failure probabilities.

The safety objective mostly consists of the quantitative safety analysis, i.e. the probability that the system may reach a critical state. Cost and performance objectives are less scope of this thesis. It is expected, that they could be expressed by simple mathematical expressions.

Optimization along a single objective often leads to trivial results. For example, the safety of a car is surely maximized if its maximum velocity is reduced to zero. However, a car that does not move is useless. On the other hand, the performance (the time it takes to travel from one point to another) is increased when the maximum velocity is increased. Thus multiple objectives need to be taken into account. In case of the car this could be the maximization of safety and (at the same time) the minimization of the time to travel from one point to another. It is clear that this does not lead to a single optimal solution. Further details on multi-objective optimization are given in Section 2.

1.4. Problem Description

Using safety analysis as an objective in multi-objective optimization includes several difficulties. On the one hand, the relation between the parameters and the safety properties is no trivial mathematical function. In an application scenario it must be
treated as a black box function. On the other hand the quantitative safety analysis is a computation time consuming task.

Genetic algorithms [3] are able to solve generic multi-objective optimization problems. However, they rely on a huge number of objective evaluation, which is not feasible with the computation time in mind. Still genetic algorithms are a rough process, which means that a good guess of the objective value might be enough to guide the algorithm in the right direction. This brings up the idea to combine an estimation algorithm with a multi-objective genetic optimization algorithm. Thus the optimization abilities of the genetic algorithm is used, and its running time is reduced even with expensive objective functions.

A widely used genetic multi-objective optimization algorithm is the non-dominated sorting evolutionary algorithm number two (NSGA-II) [4]. An estimation technique which is often used are artificial neural networks [5].

It is the scope of this thesis to analyze the combination of the NSGA-II algorithm in combination with artificial neuronal networks as an estimation algorithm for the optimization of formal specifications. SAML shall be used as formal specification language, as the SAML framework provides the necessary quantitative analysis methods.
2. Multi-Objective Optimization with Genetic Algorithms

Whenever there is a parametric system and an objective to evaluate the parameters, the system can be optimized according to the objective. In the context of formal specification a very important objective is the system safety. However maximizing the safety of a system often leads to trivial results. Considering the maximum allowed flight height of an airplane as parameter the obvious optimal result is zero. This means, if the airplane never takes off, it cannot fall down. The same applies for cars and speed limit. In both cases the system safety is maximized, but this also renders the systems useless. People who want to sell cars or airplanes might come up with another goal: “Minimization of the (system/maintenance) costs.” This however almost always affects the safety in a negative way. So to meaningfully optimize a specification several objectives must be accounted simultaneously.

This section introduces multi-objective optimization. After that an approach to solve multi-objective optimization problems with genetic algorithms is presented. Finally the non-dominated sorting genetic algorithm is explained.

2.1. Multi-Objective Optimization Problem

In real world situations one must make decisions to achieve a certain goal. The goals can also be antagonistic to each other. The parameters that might be influenced are numerically represented by decision variables. Many problems have more than only one Parameter. Thus the decision variables are denoted as a vector:

\[ x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, n > 1, x \in S \]  \hspace{1cm} (2.1)

S is called the feasible set. Unless \( S = \mathbb{R}^n \), the feasible set can add further constraints to the optimization problem. Such constraints can be upper bounds, lower bounds or discretisation of the values.

The influence of the chosen decision variables on the system and its environment is described by objective functions: \( f_1(x), f_2(x), \ldots, f_k(x) \) where \( k > 1 \) is the number of objectives. The objective functions are defined as:

\[ \forall i \in \{1 \ldots k\} : f_i : S \rightarrow \mathbb{R} \]  \hspace{1cm} (2.2)

If there are $m > 1$ objective functions they can be denoted as a vector:

$$f(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_m(x) \end{bmatrix}, m > 1$$  \hspace{1cm} (2.3)

A multi-objective optimization problem can now be defined as the problem of finding a vector of decision variables so that all objective functions are minimized:

$$\min(f) = \min \begin{bmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_m(x) \end{bmatrix}$$  \hspace{1cm} (2.4)

It is also possible to define the multi-objective optimization problem as a problem of maximization. As it is easy to turn a function that needs maximization into one that needs minimization (by negating the function) the following explanations only refer to minimization.

It is important to notice that not always a single $x \in S$ can be found, so that all objective functions $f_i$ have their minimum at the same place. This is due to the antagonistic nature of multi-objective optimization problems.

It becomes clear that the simple mathematical "<" operator is not sufficient to compare two solutions $x_a, x_b$ and decide which one is better in terms of the optimization problem. To overcome this problem, a ranking between two solutions is defined by the so called Pareto dominance.

**Definition 2.1.1 (Pareto Dominance)** A vector $u = (u_1, \ldots, u_n)$ is said to dominate another vector $v = (v_1, \ldots, v_n)$ if and only if:

$$\forall i \in \{1, \ldots, n\} : u_i \leq v_i \land \exists j \in \{1, \ldots, n\} : u_j < v_i.$$  \hspace{1cm} (2.5)

If $u$ dominates $v$ this is also denoted as $u \preceq v$.

The definition of the Pareto dominance is also illustrated in Figure 2.1, which shows the values of two objective functions $f_1$ and $f_2$. The area that is dominated by each point is the area right above the point (indicated with dashed lines). In the figure the points $P_1$ and $P_2$ are not dominated. Whereas the point $P_3$ is dominated by $P_1$ and $P_2$ is dominated by $P_4$.

If one point dominates another, the dominating point is, in terms of minimization, clearly the better one. But if there are two points which do not dominated each other ($P_1$ and $P_2$ in Figure 2.1) they are considered as equally good. This is due to the fact that none of the objective functions can be minimized further without increasing any of the other objective functions.

It becomes clear that the solution of a multi-objective optimization problem is a set of points. Thus the formal definition of the solution of a multi-objective optimization problem is given in Definition 2.1.1.
Figure 2.1.: The Pareto dominance

**Definition 2.1.1 (Pareto Optimal Set)** For a given multi-objective optimization problem, $F(x)$ with $x \in S$, the Pareto Optimal Set, $\mathcal{P}^*$, is defined as:

$$\mathcal{P}^* := \{ x \in S \ | \ \neg \exists x' \in S : f(x') \preceq f(x) \}. \quad (2.6)$$

This describes the set of all possible assignments to the decision variables so that the objective function only leads to points that do not dominate each other. This results in the following definition of the Pareto front.

**Definition 2.1.1 (Pareto Front)** For a given multi-objective optimization problem, $F(x)$ with $x \in S$, and its Pareto Optimal Set $\mathcal{P}^*$ the Pareto Front $\mathcal{PF}^*$ is defined as:

$$\mathcal{PF}^* := \{ u = f(x) \ | \ x \in \mathcal{P}^* \} \quad (2.7)$$

Spoken in natural language, this means: Every point from the Pareto optimal set leads (when being evaluated with the objective functions) to a point on the Pareto front. Each point on the Pareto front is not dominated by any of the other points on the Front. All points on the Pareto front are therefore considered as equally good (in terms of minimization).

The Pareto optimal set and the Pareto front of a multi-objective optimization problem with two decision variables $x_1, x_2$ and two objective functions $f_1, f_2$ are illustrated in Figure 2.2. It is usually no trivial task to determine the Pareto set and the Pareto front of a multi-objective optimization problem. Neither is it always possible to determine the Pareto set from the Pareto Front.

If one needs only one best solution, further criteria are required. These criteria are usually domain specific and/or based on the knowledge of experts. In most
cases these objective functions are antagonistic to each other. Thus the Pareto front with all optimal system configurations can be calculated. But then one wants to build exactly one specific system rather than a set of systems. The problem is that in such a case no mathematical tool can give the answer to the question which one to build. So an expert from the domain is needed to pick one specific point from the Pareto front which, to his knowledge, is the best solution. Of course one can specify further rules how to determine one single best solution, but these are then very domain and/or application specific and not considered within the scope of this thesis.

Solving a multi-objective optimization problem means finding the Pareto optimal set. Solving real world multi-objective optimization problems can add further difficulties to the optimization task:

- The decision variables may have upper and/or lower bounds. This clearly is the case for almost any parameter in technical systems like time, voltage, current, light intensity, velocity, acceleration, temperature, etc. Maybe this also applies to financial systems even though the economists often expect infinite growing.

- The decision variables can be of multiple different types, such as discrete or real valued. Computer systems always work with discrete values. Also a decision variable that describes the selection of one part out of a group of similar parts can only referee to one or another, but nothing in between.

- No further information about the objective functions like the derivative is given nor can it be easily calculated.
The evaluation of an objective function can be very expensive. This can either be the case if an objective needs enormous computation power or if the function is represented by a technical process that cannot be expressed and calculated in a mathematical way. The later one can be the case if the objective function is given as a laboratory experiment.

There are several approaches for solving multi-objective optimization problems. Among the more famous ones there are simulated annealing [6] and particle swarm simulation [7] and genetic algorithms. For this thesis the genetic approach has been chosen. The next section will introduce an approach to solve generic multi-objective optimization problems by the usage of genetic algorithms.

2.2. Multi-Objective Evolutionary Algorithm

2.2.1. Origin of Evolutionary Algorithms

The basic idea behind evolutionary algorithms is the adaption of the evolutionary process known from genetics. Hence, good solutions are selected from a set of arbitrary solutions. Then the selected solutions are altered to gain better solutions. This process is repeated until good enough solutions are found [8, 9]. The whole procedure follows the schema of evolution in biology. Due to this origin, the terminology used in the evolutionary algorithm domain is adopted from the genetic domain.

The blueprint of an individual is encoded as a sequence of genes, called chromosome. The shape of the organism belonging to a specific genotype is expressed as phenotype. A set of individuals are called a population. A population can be further divided in generations, so that all individuals in one generation were inherited from the prior generation. The evolutionary process consists of the repeated application of three different operations:

Selection refers to the process of selecting two individuals for the creation of a new individual. In the biological context this is often influenced by some kind of fitness of the phenotype compared to other individuals of the same population.

Crossover combines the genes from two (or more) parents into one child. Thus a new chromosome is created by combining parts of two given chromosomes.

Mutation refers to the random modification of a chromosome.

The application of the selection, crossover and mutation operators on a given population is depicted in Figure 2.3. After a new offspring population is generated, they become the current population. This loop is processed repeatedly. Successive generated populations are termed generations.
Fitter individuals are expected to survive this process. This leads to an inheritance of the genes of fitter individuals to the next generation. As a result the individuals are expected to converge to the optimum in terms of the fitness criteria. Due to the mutation also entirely new individuals are invented. This maintains a certain spread of differing individuals among the population. The result of this process is basically what one expects from an optimization algorithm.

2.2.2. Evolutionary Algorithms for Optimization

A multi-objective evolutionary optimization algorithm consequently applies the genetic evolution strategy to solve a given optimization problem. From a more technical point of view this process can also be seen as a guided search towards the Pareto front.

The evolutionary process maps in the following way to the multi-objective optimization problem:

- The vector of decision variables \( (x) \) is treated as the chromosome. Each entry in the vector maps to one gene or a partial sequence of genes. Consequently \( x \) can be called an individual.

- A fitness function is required to evaluate the individuals. In terms of the
multi-objective optimization problem it is straightforward to use the objective functions as fitness criteria.

- The results of the objective function evaluation become the phenotype.
- Use the Pareto dominance operator \( \preceq \) to compare the fitness of two individuals.

This of course leads to some questions: How are the decision variables encoded. What are the semantics of the operators (selection, crossover and mutation) for the numerically defined chromosome data type.

2.2.3. Encoding of the Chromosome

There are different approaches to encode the decision variables and the genotype into the chromosome. Common variants are binary coded algorithms and real coded algorithms.

**Binary Coding**

The first evolutionary algorithms that were developed adopted the chromosome encoding scheme from biology. All decision variables were turned into a binary representation and the results were concatenated to one binary string.

Discrete integer numbers can easily be encoded in a binary string. Common encoding schemata are binary coded decimal or gray coding. In any case an integer number is transformed into a sequence of two different symbols. This closely relates to the four valued encoding of the chromosome in biology.

**Real Coding**

The binary encoding of the chromosome performs well, as long as only discrete values are used in the chromosome. Many optimization problems, however, require real valued decision variables. Even though there are binary representations for real values (float and fixed point) these are not used. In so called real coded genetic algorithms a vector of real valued variables is used as a chromosome. This of course differs from the biological prototype. However, it fits the requirements of many real world optimization problem requirements.

2.2.4. Classification of Evolutionary Algorithms

The basic idea of genetic algorithms is to select solutions and alter them to create new (and hopefully better) solutions. This procedure can be realized in certain different variants. Two obvious approaches are a generational approach and a steady state approach. Another aspect that has to be taken into account, is whether elitist solutions may vanish or not.
Generational Algorithms

The generational approach creates a whole new set of individuals (the generation) for every iteration. For every individual in the offspring population the parents are selected and crossed and the becoming children are then mutated. Afterwards the original and the offspring generation are combined and a new population is extracted to form the next generation.

Steady State Algorithms

The steady state approach only creates one or two new individuals in every iteration. Thereto two parents are selected. The crossover and mutation operator is applied on them. Afterwards the newly created individuals replace the worst individuals in the population. Thus only a small part of the population changes in every iteration.

Elitism

Elitism refers to an approach that preserves (previously found) best solutions. This implies that elitist individuals can directly pass from one generation to the next generation instead of only inheriting their elitist genes to the next generation.

Regarding the fact that there is no single best solution for a multi-objective optimization problem the list of preserved solutions can grow infinitely. In practical realizations a compromise has to be found between preserving every elitist solution and maintaining only a limited number of solutions.

However, to preserve a good diversity among the population, it can be an advantage to evolve some less fit individuals.

2.3. Crossover Operators

The crossover operation takes two (or in some cases even more) individuals and combines them to several new individuals. The initial individuals are called parents and the resulting individuals are the children. The intention of the crossover is to combine the properties from the parent and thus create children that share the properties from both parents. So if there are two fit parents, the children are expected to be even fitter.

According to the encoding of the chromosomes, different crossover operators are available. A widely spread crossover operator for binary coded algorithms is the single point crossover operator. The simulated binary crossover operator is a statistical approach to emulate the single point crossover operator for real coded genetic algorithms.
2.3.1. Single Point Crossover

The crossover of two such binary strings is illustrated in Figure 2.4. At a certain position the binary strings are split and the tails of both chromosomes are exchanged. The position of the crossover point is randomly chosen. The single point crossover consequently emulates the biological crossover. Variants of this approach are within the encoding of the values into binary representation or the number of the crossover points.

\[
\begin{array}{c}
\begin{array}{cccccccc}
0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
\end{array} \\
\begin{array}{cccccccc}
1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
\end{array}
\end{array}
\times
\begin{array}{c}
\begin{array}{cccccccc}
1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\
\end{array} \\
\begin{array}{cccccccc}
0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\
\end{array}
\end{array}
\]

Figure 2.4.: One-point crossover of two binary strings

2.3.2. Simulated Binary Crossover Operator

The single point crossover operator cannot be applied for real coded genetic algorithms. This is because there is no natural crossover point in a real number, as there is in binary coded numbers.

The simulated binary crossover operator is a crossover operator to overcome this limitation of real coded numbers [10, 11]. Its intention is to create the children so that their probability distribution equals the distribution from a single point crossover for binary coded numbers. Additionally the simulated binary crossover operator is designed in a way, so that children are always placed symmetrically around the parents. This means, that none of the parents is favored.

For the calculation of the children \( c_1 \) and \( c_2 \) from the parents \( p_1 \) and \( p_2 \) a spread factor is defined:

\[
\beta = \left| \frac{c_1 - c_2}{p_1 - p_2} \right|
\]  

(2.8)

The relation between the spread factor \( \beta \) and the location of the children in relation to the position of the parents is illustrated in Figure 2.5. A spread factor less than one indicates that the children are located between both parents. A spread factor greater than one indicates children outside the interval between both parents.

For the spread factor \( \beta \) from Equation (2.8) the following distribution is defined:

\[
P(\beta) = \begin{cases} 
\frac{1}{2} (\eta_c + 1) \beta^{\eta_c}, & \text{if } \beta \leq 1 \\
\frac{1}{2} (\eta_c + 1) \frac{1}{\beta^{\eta_c+2}}, & \text{else}
\end{cases}
\]  

(2.9)

This distribution is selected so that it is close to the distribution of the spread factor if a single point crossover on binary coded numbers would be used. Further details on the derivation of the distribution are given in the publication about the simulated binary crossover operator [10].
The parameter $\eta_c$, with $\eta_c > 0$, is used to control the spreading of the children. With a smaller $\eta_c$ value, the distribution allows children further away from the parents. A greater value of $\eta_c$ leads to children closer to the parents. Figure 2.6 shows the probability distribution of $\beta$ for two distinct values of $\eta_c$ according to the distribution in equation (2.9).

\[
\beta_r = \begin{cases} 
(2r)^{\frac{1}{\eta_c}} & \text{if } r \leq \frac{1}{2} \\
\left(\frac{1}{2(1-r)}\right)^{\frac{1}{\eta_c}} & \text{else.} 
\end{cases} \tag{2.10}
\]
Afterwards, the spread factor is used to calculate the two children:

\[ c_1 = \frac{1}{2}((1 + \beta_r)p_1 + (1 - \beta_r)p_2) \]  \hspace{1cm} (2.11)

\[ c_2 = \frac{1}{2}((1 - \beta_r)p_1 + (1 + \beta_r)p_2) \]  \hspace{1cm} (2.12)

The probability distributions for two children derived from one pair of parents are shown in Figure 2.7. In the plotted distribution the parents have the values three and seven. The locations of the parents are denoted by vertical dashed lines. The figure shows the distribution for two \( \eta_c \) values (\( \eta_c = 2 \) and \( \eta_c = 5 \)).

![Figure 2.7: Distribution for two children generated from a pair of parents, with the values three and seven](image)

Another interesting fact about the simulated binary crossover is, that the operator avoids children close to the mean value of both parents. This can be easily seen in Figure 2.7. The probability density of both children is zero at the mean value between both parents.

### 2.4. Non-Dominated Sorting Genetic Algorithm II

This section covers the NSGA-II (Non-dominated Sorting Genetic Algorithm II) algorithm. NSGA-II is an elitist generational evolutionary optimization algorithm.
In the first subsection the algorithm itself is explained. Afterwards a variant, used for this thesis, is presented.

The NSGA-II algorithm was chosen because it performed well not only on several test problems but also on real world problems [12, 13, 4]. Further on there is a successful application of the NSGA-II algorithm together with an artificial network as optimizer on a few test problems [14]. Thus it is expected that the combination may also perform well for the optimization of formal specifications.

Algorithm

NSGA-II is a genetic multi-objective optimization algorithm. It was introduced in 2002 by Deb et al. This section gives a short introduction to NSGA-II. For further details see their publication [4].

Figure 2.8.: General idea of non-dominated sorting

The NSGA-II algorithm relies on a sorting algorithm named fast non-dominated sorting. This algorithm sorts a given set of points according to their Pareto dominance. All points that are not dominated by any other point belong to the first front. All points which are dominated only by points from the first front belong to the second front. The third front consists of points that are dominated only by points from the first and the second front. This goes on until all points are assigned to a front. The total number of resulting fronts of course depends on the set of points that are sorted.

Figure 2.8 illustrates the general idea of non-dominated sorting. The points belonging to the first front are depicted by a “x” sign. The points of the second front by a “+” and the third front with a “o” signs. It is important to mention that there is no order within each front. According to the Pareto dominance none of the points in one front dominates any other in the same front. Thus they are all
considered equally optimal. A detailed description of the non-dominated sorting algorithm is given in the publication about the NSGA-II algorithm [4].

To maintain a good diversity among the Pareto optimal solutions the NSGA-II algorithm introduces the crowding distance as additional optimality criteria. The crowding distance is calculated for each point in a non-dominated set and estimates the density of the solutions surrounding this point. To determine the crowding distance, the normalized distance between the last and the next point along every objective is calculated and summed up. With the distances depicted in Figure 2.9 the crowding distance for point $i$ out of $k > 0$ non-dominated points calculates as follows:

$$i_{\text{distance}} = \sum_k \frac{d_i^k}{d_{\text{max}}^k}, \quad 0 < i < k$$

(2.13)

Informally spoken, a smaller value of $i_{\text{distance}}$ indicates that a point is located in a more crowded region. This enables the algorithm to prefer points in a less crowded region over points in a more crowded region. This guides the algorithm to a good diversity in the population.

The crowding distance of the outermost points cannot be calculated, as there are no next points to calculate the distance. To keep the maximum diversity among all solutions, the crowding distance of the outermost points in a front is defined as infinite.

To compare two arbitrary points the crowded-comparison operator ($\prec_n$) is introduced. It compares two individuals according to their Pareto rank and their crowding distance. First of all, the Pareto rank is compared. Only if two individuals have the same Pareto rank, they are compared by their crowding-distance.
Definition 2.4.1 (Crowded-Comparison Operator) If $i_{\text{rank}}$ denotes the Pareto rank (by how many other individuals a point is dominated) and $i_{\text{distance}}$ denotes the crowding distance of point $i$, then the crowded-comparison operator ($\prec_n$) is defined as follows:

$$i \prec_n j \Leftrightarrow (i_{\text{rank}} < j_{\text{rank}}) \lor (i_{\text{rank}} = j_{\text{rank}} \land i_{\text{distance}} > j_{\text{distance}})$$ (2.14)

The NSGA-II main algorithm is listed in Algorithm 1. The make-new-population method performs the selection, crossover and mutation operations. After the initial population and the first offspring population is created, the algorithm processes $u > 0$ generations in a loop. The first operation in the loop unites the current population and the former offspring population. All individuals are then sorted according to their domination. After that the next generation is extracted. This is done as follows: As long as there is space in the next generation set, add whole fronts starting with the first one. By the time no complete next front fits into the next population, this particular front is sorted according to the crowding distance. Then the next population is filled up with individuals from a less crowded region. Finally a new offspring population is created and the loop starts again.

**Algorithm 1 NSGA-II Main Algorithm**

\begin{verbatim}
$P_0 = \text{create-initial-population()}$
$Q_t = \text{make-new-population}(P_0)$

while $t < u$ do
  $R_t = P_t \cup Q_t$
  $\mathcal{F} = \text{fast-non-dominated-sort}(R_t)$
  $P_{t+1} = \emptyset$
  $i = 1$
  \begin{verbatim}
  while $|P_{t+1}| + |\mathcal{F}_i| \leq N$ do
    $\text{crowding-distance-assignment}(\mathcal{F}_i)$
    $P_{t+1} = P_{t+1} \cup \mathcal{F}_i$
    $i = i + 1$
  end while
  $\text{Sort}(\mathcal{F}_i, \prec_n)$
  $P_{t+1} = P_{t+1} \cup \mathcal{F}_i[1 : (|P_{t+1}|)]$
  $Q_{t+1} = \text{make-new-pop}(P_{t+1})$
  $t = t + 1$
end while
\end{verbatim}

The extraction of the next generation (within the outer while loop) is also depicted in Figure 2.10. In the illustration the fronts one to three entirely fit in the next population. Front number four, however is sorted according to the crowding distance and only the lesser crowded points became part of the next generation. The more crowded points of front four and the whole fronts five and six are dropped.
The authors of the original NSGA-II algorithm proposed to use the simulated binary crossover (SBX) [10] operator and a polynomial mutation \(^1\) operator.

**A Variant of the algorithm**

In this thesis the NSGA-II algorithm is used to optimize multi-objective optimization problems that contain numerically expensive functions. For the algorithm to terminate in a feasible time the overall number of function evaluations must be kept at a low level. This could be reached by reducing the number of generations. However a feasible number of generations is required for the evolutionary algorithm to work properly. Another approach is to reduce the size of the population. In the original NSGA-II algorithm, this directly influences the maximal number of optimal results that can be found by the algorithm. If the first front gets larger than the population, only the less crowded points survive. This however, includes that some non-dominated points are dropped. Thus a compromise needs to be found between population size and the number of generations. Nevertheless there might have been some good solutions that were dropped throughout the algorithm.

To relax the problem of possible dropped good individuals an external storage is introduced [15]. This storage holds all individuals that were evaluated throughout the optimization process. In contrast to the archiving approach proposed by Gaspar-Cunha and Vieira [15], the individuals are only stored in the external storage and not integrated in the active optimization process. After the algorithm terminates (after a determined number of generations) the external storage is sorted using the non-dominated sorting algorithm. The first front of this

---

\(^1\)taken from the NSGA-II source code version 1.1.5:  
Figure 2.11.: Solutions that are dropped by the NSGA-II algorithm are preserved in the external storage.

The final sorting process is considered as the final result of the algorithm. The external storage is not part of the original NSGA-II algorithm.
3. Safety Optimization

3.1. Safety Analysis

Many technical systems are reactive systems. Such systems are influenced by their environment and in return they also influence their environment. This is illustrated as block diagram in Figure 3.1.

![Block diagram of a reactive system that is interacting with the environment](image)

Figure 3.1.: Block diagram of a reactive system that is interacting with the environment

It is hard to draw a precise border between the system and its environment. In practice the system usually is a closed device with electrical and/or mechanical connections, such as a motor or a computer that runs a certain program. For analysis though, the border is often drawn a bit differently. The system often only consists of the control algorithm. It operates on the sensor data and generates appropriate actor control data. The rest is part of the environment, that takes actor control data as input, and generates the according sensor data as output. Thereto the environment model consists of three different parts. First of all there is a model of the actors. Secondly, there is a model of physical properties, such as motion equations or gravity law and thirdly, the state of the physical properties is turned into sensor data by a model of the sensors.

As mentioned the precise separation in the two domains depends on the analysis case. Anyhow the important point is that each part depends on the output of the other one. Thus they form a feedback loop. As a consequence, reactive systems cannot be analyzed by feeding arbitrary input data to the input and observing the output of the system. The system can only be analyzed in combination with a valid environment.
A very important property of reactive systems is, that they usually do not terminate. They operate in an endless loop and never reach a final state. A very intuitive example is an adaptive cruise control in an automotive application. The cruise control systems continuously measures the distance to the car in front of the vehicle. The vehicle will accelerate if the distance increases or decelerate if the distance decreases. In this intuitive process there is no termination condition\(^1\). Analyzing reactive systems leads to various kinds of difficulties:

- Reactive systems cannot be described and analyzed in terms of Hoare logic. Due to the fact that the system never terminates, there are no post conditions.

- As soon as the system has an internal state, the output of the system does not only depend on the current input, but also on the previous input values. This leads to the question what are valid input data and how to can sound test values be created. Apparently a (over the time) constant distance measure is a valid input to the cruise control system, but is this realistic. In contrast to a constant value an arbitrarily changing distance value might not be valid input data, because cars do not teleport from one position to another.

- The behavior of the environment of the system does not necessarily have a deterministic behavior. In case of the cruise control, the car in front of the vehicle is controlled by a human driver and thus may accelerate or decelerate arbitrarily.

- In many cases it is a priori known that a system can fail under certain circumstances. A wrong sensor reading for example may lead the cruise control into a crash. Nevertheless, a single sensor failure must not lead to a crash. The interesting remaining question is, if the occurrence probability of all the atomic failures (i.e. the sensor failure) is known, what the overall failure probability is.

For the first of the mentioned difficulties, reactive systems can be specified in terms of computation tree logic formulae. These enable the designer to specify the evolution of a signal over time and especially also to describe infinite time periods. This can be done as long as discrete time steps are used. Almost all computer systems are based on discrete time steps. Analogue systems can often be described in a time discrete manner up to a certain accuracy that is enough in the context of the analysis. Model checking is a technique to analyze whether a state system fulfills a certain computation tree logic formulae or not. Model checking can answer questions about an infinite running time in a finite time duration.

This still does not solve the question about valid and sound input values. A widely used approach is to implement a model of the environment. For the cruise control system the environment model will contain the equations of movement up

\(^1\)Of course the driver of the car can decide to disable the system, but otherwise the cruise control should never stop.
to a certain accuracy. In a basic version this will be a certain maximum velocity and maximum acceleration. A more sophisticated model may also consider the friction or special behavior of a car in turns and on a straight road or certain weather conditions.

Environment models often contain non deterministic behavior (i.e. the driver who will accelerate or decelerate the car). Thus a framework that allows for the modeling of environmental systems must support non deterministic behavior. Non determinism is supported by many model checkers, in a way so that the worse case is considered.

The first three points consider the qualitative behavior of the system. The result of qualitative analysis always points out if a specification holds for a given system, or points out an example under which the specification is violated.

The problem last mentioned, targets the probabilistic domain. The model of the cruise control system can be extended in such a way, that the failure of the distance sensor becomes a part of the model. The occurrence of the failures is controlled by a probability distribution. It then becomes clear that the system can fail (at least if the sensor fails). The question is how likely it is that the complete system will fail. In case of a very trivial system, the question about the failure probability might be trivial. However, it becomes more difficult with the rising complexity of the system. For example if the cruise control consists of more than one sensor, or if it is able to detect certain failure patterns and reacts appropriately. Probabilistic analysis of transition systems can be performed with probabilistic model checkers.

The last point relates to the quantitative properties of the system (i.e. if a system can fail). The answer to a quantitative analysis of a systems gives the probability that a given specification holds or does not hold for a certain system.

There are certain model checkers to deal with probabilistic behavior of models. Rather than plain non deterministic decisions, a probability distribution over all possible choices must be specified in the model. If, for example, the failure modes of all system components are given, the analysis can calculate the overall failure probability.

At this point it appears that a broad analysis of a system requires both, qualitative and quantitative aspects. As pointed out earlier, special tools for both domains exist. All of these tools expect the models that are to be analyzed in their own dedicated input language. Thus it is very difficult to ensure that the semantic of the models in both domains (of the same system) equals.

3.2. Safety Analysis Modeling Language

To fill up the gap between qualitative and quantitative model based analysis the safety analysis modeling language (SAML) has been developed by Güdemann et. al [1]. This section introduces the SAML language and how it fits into the optimization problem.

A SAML model describes a set of finite state automata. All automata are
executed in synchronous discrete time steps. Thus SAML is sufficient to model all
time and value discrete systems with a finite state space. This of course applies for
all computer systems, and many other technical systems. However, systems that
are commonly described by differential equations cannot be fully implemented in
SAML. In such cases time or value continuous system can often be approximated
by discrete systems up to a sufficient accuracy. If the discrete approximation of a
time or value continuous system is accurate enough is a difficult question, but is
not scope of this thesis.

3.2.1. SAML Syntax
This section introduces the syntax of the safety analysis modeling language. Figure
3.2 lists a simplified version of the SAML grammar in extended Backus-Naur
form [1]. A model may consist of zero or more formulae and constants. A constant
is defined by a type, a name and a value. It can be of three different types: int,
float or double. A constant can be used at every place where the corresponding
numerical value is allowed. A formula consists of a name and a prepositional expres-
sion. Every formula must evaluate to a Boolean value, i.e. to true or false. They
can be used as abbreviations in the activation conditions of the state transition
rules.

The state automatons itself are grouped into so called modules. A module starts
with the keyword module and ends with endmodule. Every module has one or more
state variables that are defined for a bounded range of integer values. Every state
variable has one certain start value. After the definition of the state variables, a
set of transition rules follows. The transition rules describe under which conditions
the module may change into which of the possible next states. A transition rule
consists of an activation condition and a list of non deterministic choices. The
activation condition is given as formulae of propositional logic and states when this
rule is active. Each choice starts with the keyword choice and an expression for
the next state in brackets. Several choices are concatenated with the plus (+) sign
and denote the non deterministic behavior of the system.

Within each choice a discrete probability distribution over the next state assign-
ments is expressed. Thereto a probability is followed by a : and an assignment to
the state variable in the next state. The state variable in the next state is denoted
by the name of the state variable which is followed by an apostrophe ('). Again,
several next state assignments with different probabilities can be concatenated with
the plus (+) sign. In a valid model the probabilities in one choice must sum to one.

Pure deterministic behavior is expressed with only one choice and a probability
distribution with only one value and a probability of one. Non determinism is spec-
ified by adding several choices, that all have a single valued probability distribution
with the probability of one. Strict probabilistic behavior is modeled with only one
choice and a certain probability distribution.

If there is more than one state variable in a module, several next state assignments
can be concatenated with the ampersand (&) symbol. All assignments to the state
Figure 3.2.: A simplified version of the SAML syntax in extended Backus-Naur form variables are executed in parallel. It is important that all state variables get a next state value assigned in every choice and under every probability. If a state variable explicitly does not change, this can be expressed by assigning the current state value to the next state (i.e. \( \text{state}_\text{var}' = \text{state}_\text{var} \)).

The update rules of one module are not allowed to update the state variables of another module. However the value in current state of all variables is globally
readable. Thus the behavior of one module may depend on the state of another module.

A very simple example for a SAML model is listed in Figure 3.3. The Example shows two modules. The battery module represents a counter that counts from 10 to 0. It has a strict deterministic behavior. The other module is the mydevice module. This one has probabilistic and non-deterministic behavior. The device has the three states: failure (−1), off (0) and on (1). Whenever the device reaches the error state, it stays there. As long as the battery counter is greater than zero, the device can arbitrarily turned on and off. In this case it can also fail with a probability that is specified with the constant e_device. If the battery counter is zero and the device is not in the error state it turns off.

\[
\text{constant double e_device := 0.001;}
\]

\[
\text{module battery}
\]
\[
\text{battery_state : [0..10] init 10;}
\]
\[
\text{battery_state > 0 -> choice:(1: (battery_state' = battery_state - 1));}
\]
\[
\text{battery_state = 0 -> choice:(1: (battery_state' = battery_state))};
\]
\[
\text{endmodule}
\]

\[
\text{module mydevice}
\]
\[
\text{// -1 = error; 0 = off; 1 = on}
\]
\[
\text{device_state : [-1..1] init 0;}
\]
\[
\text{device_state = -1 -> choice:(1: (device_state' = - 1));}
\]
\[
\text{battery_state > 0 & (device_state = 0 | device_state = 1) ->}
\]
\[
\text{choice:(e_device: (device_state' = -1) +}
\]
\[
\text{1-e_device: (device_state' = 1))} +
\]
\[
\text{choice:(e_device: (device_state' = -1) +}
\]
\[
\text{1-e_device: (device_state' = 0))} +
\]
\[
\text{battery_state = 0 & !(device_state = -1) ->}
\]
\[
\text{choice: (1: (device_state' = 0));}
\]
\[
\text{endmodule}
\]

Figure 3.3.: A simple SAML model

3.2.2. SAML Semantics

The semantic of a SAML model is, in the first place, defined for a model with only one single module with one state variable. Afterwards it is shown how a SAML
model with more than one module can be expressed as a single module as well.

A Markov decision process (Definition 3.2.1) is a transition system with a finite set of states. Each state is labeled with a set of atomic propositions. A Markov decision process allows for the combination of probabilistic and non deterministic transitions [16].

**Definition 3.2.1 (Markov Decision Process)** A Markov decision process is a tuple

\[
\tau_{MDP} = (S, \text{Steps}, \text{AP}, L, s_0),
\]

where \( S \) is a finite set of states, \( \text{Steps} : S \to 2^{\text{Dist}(S)} \) is a non empty set of probability distribution functions, which assign to each state a finite non empty set of probability distributions, \( \text{Dist}(S) \), over the successor states, \( \text{AP} \) is a finite set of atomic propositions, \( L : S \to 2^{\text{AP}} \) is a labeling function, that labels each state \( s \in S \) with a subset of \( \text{AP} \) and an initial state \( s_0 \in S \).

It is important to notice that the Markov decision process assigns a set of probability distributions over the successor states to each state. Thereby the non deterministic behavior in combination with the non determinism is expressed. Each probability distribution denotes the probability by which the system switches to a certain successor step.

With the given definition of a Markov decision process, a SAML model maps to the Markov decision process in the following way:

- Every possible value for the state variable is defined as a state \( s \in S \).
- The initial value of the state variable represents the initial state \( s_0 \).
- The update rules in the SAML model denote the possible transitions of the Markov decision process. The activation condition of the update rule is true for a subset \( S' \subseteq S \) of all states. For each of the states the \( \text{Steps} \) function gives a set of probability distributions over the next states. This set of probability distribution functions is covered by the list of non deterministic choices. Within each choice a certain distribution over the successor states is specified.

This still leaves room for ill formed models. Thus there are further constraints that must hold for a SAML model to be a valid model:

- For all non deterministic choices, all probabilities \( -p_i \) must sum to one: \( \sum_i p_i \equiv 1 \). This ensures that the given distribution is a valid probability distribution.
- For each reachable state one activation condition must evaluate to true: \( \forall s \in S : \bigvee_i \phi_i \equiv true \).
- For each state there must only be one activation condition that evaluates to true: \( \forall s \in S : \forall i \neq j : \phi_i \land \phi_j = false \).  

27
The last two constraints ensure that there is exactly one active update rule in every state. This also ensures that for every state a next state is defined.

This far the semantics of a SAML model with one single module was defined. A complete SAML model can consist of several modules:

**Definition 3.2.1 (A SAML Model)** A SAML model is defined as a tuple

\[ S = (C, \mathcal{F}, \mathcal{M}), \]

where \( C \) is a set of constants, \( \mathcal{F} \) is a set of formulae and \( \mathcal{M} \) is a set of modules.

With the parallel composition of SAML modules in Definition 3.2.1, a multi module model can be expressed in a single module. The complete model in a single module, \( M \), consisting of the modules \( M_1, \ldots, M_n \) is created by a parallel composition of the specific modules:

\[ M := M_1 || \ldots || M_n \]

**Definition 3.2.1 (Parallel composition of SAML modules)** With

\[ \phi^i_{o} \rightarrow \sum_{k=1}^{c} \text{choice}^i_{ok} \left( \sum_{l=1}^{d_k} p^i_{okl} : u^i_{okl} \right) \]

denoting the \( o \)-th update rule (with \( c \) non deterministic choices and a probability distribution with \( d_k \) elements in the \( k \)-th choice) out of \( O \) update rules in module \( i \), and

\[ \phi^j_{p} \rightarrow \sum_{m=1}^{e} \text{choice}^j_{pm} \left( \sum_{n=1}^{f_m} p^j_{pmn} : u^j_{pmn} \right) \]

denoting the \( p \)-th update rule (with \( e \) non deterministic choices and a probability distribution with \( f_m \) elements in the \( m \)-th choice) out of \( P \) update rules in module \( j \), the update rules of the parallel composition of two modules, \( M = M_i || M_j \) is defined as:

\[ \forall o \in \{1, \ldots, O\} : \forall p \in \{1, \ldots, P\} : \]

\[ \phi^i_{o} \land \phi^j_{p} \rightarrow \sum_{k=1}^{c} \sum_{m=1}^{e} \text{choice} \left( \sum_{l=1}^{d_k} \sum_{n=1}^{f_m} \left( p^i_{okl} \cdot p^j_{pmn} : u^i_{okl} \land u^j_{pmn} \right) \right). \]

The remaining module \( M \) then contains the state variables of all partial modules. The activation condition of the update rules in the remaining module are the conjunction of all possible combinations of the activation conditions of the partial modules. The non deterministic updates for each of the update rules in the new module consist of the combination of the non deterministic updates from the partial modules that were active for the stated activation condition.

The probability distribution in every choice of the composed module is the combination of the according probability distributions of the partial modules. The update assignments in the new probability distribution are the parallel assignments of the according assignments from the partial modules. An extensive example of the parallel assignment of two simple modules is presented in [1].
3.2.3. Analysis of a SAML Model

As pointed out a SAML model covers probabilistic and non-deterministic behavior. Thus it can be used for qualitative and quantitative analysis. Nevertheless no model checker exists that can read SAML models directly. Thus there are a number of syntactic model transformations to transform a SAML model into the input language of certain model checkers [1]. One transformation is into the input language of the NuSMV\(^2\) [17] model checker for qualitative analysis. Another transformation exists to turn a SAML model into a representation that can be processed by the PRISM\(^3\) [18] model checker. PRISM is a tool for qualitative model based analysis.

Both transformations have a well defined semantic. However, the transformations itself are out of the scope of this thesis. The interested reader is redirected to the SAML relevant literature [1]. The qualitative analysis in this thesis relies on the Java implementation of the transformation into the PRISM input language.

3.3. Optimizable Safety Analysis Modeling Language

A SAML model describes the behavior of a system in a formal way. This means the description is clear without any ambiguity. Thus the model can also be treated as a specification of a system. Due to the fact that it is then an invariant specification there is no way to optimize a SAML model. To overcome the invariance of the safety analysis modeling language, well chosen variation points are introduced in the optimizable safety analysis modeling language (O-SAML). These variation points are realized in terms of parameters.

There are parametric constants, i.e. constants whose value is variable at the design time. These are referred as \textit{param constants}. Another variation ability is the expression of alternative modules. Due to the design time, one or another module can be chosen out of a group of alternative modules. The groups of modules are referred as \textit{param modules}.

3.3.1. Syntax

A very simple version of the example from Figure 3.3 with two parameters is listed in Figure 3.4. First, the constant \texttt{e_device} is now a parameter. Second, the model contains the two interchangeable modules \texttt{battery_strong} and \texttt{battery_weak}. The difference between the two battery modules lies in the update rule which discharges the battery. In the strong battery, the update counts in single steps and in the weak version, it counts with double steps.

Although the term “param constant” might sound contrary, it makes sense in the way that the parameters are used to describe a set of models. In a specific model,


\(^3\)18. December 2011: http://www.prismmodelchecker.org/
param constant double e_device := 0.001;

param module battery
    module battery_strong
        battery_state : [0..10] init 10;
        battery_state > 0 ->
            choice:(1: (battery_state’ = battery_state - 1));
        battery_state = 0 ->
            choice:(1: (battery_state’ = battery_state));
    endmodule

    module battery_weak
        battery_state : [0..10] init 10;
        battery_state > 0 ->
            choice:(1: (battery_state’ = battery_state - 2));
        battery_state = 0 ->
            choice:(1: (battery_state’ = battery_state));
    endmodule
endparam

module mydevice
    // -1 = error; 0 = off; 1 = on
    device_state : [-1..1] init 0;
    device_state = -1 -> choice:(1: (device_state’ = - 1));
    battery_state > 0 & (device_state = 0 | device_state = 1) ->
        choice:(e_device: (device_state’ = -1) +
        1-e_device:(device_state’ = 1)) +
        choice:(e_device: (device_state’ = -1) +
        1-e_device:(device_state’ = 0)) +
    battery_state = 0 & !(device_state = -1) ->
        choice: (1: (device_state’ = 0));
endmodule

Figure 3.4.: A simple SAML model with parameters
out of this set each parameter gets a specific value and thus is constant within the specific model.

The grammar for the optimization extended version of SAML is listed in Figure 3.5. The O-SAML grammar adds the rules `metamodule`, `metaconstant`, `paramconst` and `parammodule` to the SAML grammar.

A parameter constant is similar to the definition of a constant. The difference lies in the value that is assigned to the constant. For the parameter constant the value is a range not a specific value.

The parameter module is a container around a set of alternative modules. These alternative modules are intended to model similar system components, with only slightly different behavior. This is the case for different types of batteries (as used in the example) but also for different sensor types which are used for the same measure (such as a laser based distance sensor and an ultrasonic based distance sensor).

### 3.3.2. Semantics

An O-SAML model is defined similarly to a SAML model, as tuple:

**Definition 3.3.1** A O-SAML model is defined as a tuple

\[
OS = (C^O, PC^O, F^O, M^O, PM^O),
\]

where \(C^O\) is a set of constants, \(F^O\) is a set of formulae, \(M^O\) is a set of modules, \(PC^O\) is a set of sets of constants (representing the parameter constants) and \(PM^O\) is a set of sets of modules (representing the parameter modules).

Since it is not required to formally analyze a O-SAML model the semantic of a O-SAML model is defined with respect to a SAML model. A O-SAML model covers a family of SAML models. A specific SAML model out of a variant O-SAML model is created by instantiating the O-SAML model. An instance of a O-SAML model thus is one specific member out of the O-SAML model family:

**Definition 3.3.1 (Instance of a O-SAML model)** A SAML model

\[
O = (C, F, M)
\]

is an instance of a O-SAML model

\[
OS = (C^O, PC^O, F^O, M^O, PM^O)
\]
if and only if

\[
\begin{align*}
\forall f \in \mathcal{F}^O : & \quad f \in \mathcal{F} \\
\forall f \in \mathcal{F} : & \quad f \in \mathcal{F}^O \\
\forall c \in \mathcal{C}^O : & \quad c \in \mathcal{C} \\
\forall pc \in \mathcal{PC}^O : & \quad \exists c \in pc : c \in \mathcal{C} \\
\forall c \in \mathcal{C} : & \quad c \in \mathcal{C}^O \lor \exists pc \in \mathcal{PC}^O : c \in pc \\
\forall m \in \mathcal{MO} : & \quad m \in \mathcal{M} \\
\forall pm \in \mathcal{PM}^O : & \quad \exists m \in pm : m \in \mathcal{M} \land \neg \exists m' \in pm : m \neq m' \land m \in \mathcal{M} \\
\forall m \in \mathcal{M} : & \quad m \in \mathcal{MO} \lor \exists pm \in \mathcal{PM}^O : m \in pm.
\end{align*}
\]

(3.10)

Here \( \mathcal{PC}^O \) is a set of sets of values. The outer set covers all param constants in the model. The inner sets describe the available values for each of the parameter constants, which depend on the type of the parameter constant. Similar to this is \( \mathcal{PM}^O \) a set of sets of modules. Again the outer set holds all param modules and the inner set holds the modules in each of the parameter module.

Instantiating a O-SAML model means to choose a specific module for each set of modules in \( \mathcal{MP}^O \) and choosing a specific value for each parameter constant in \( \mathcal{PC}^O \). All constants, formulae and modules from the O-SAML model remain in the instance. It is important that for every parameter module only one module can be chosen.

3.3.3. Analysis of a O-SAML Model

The weak definition of the semantics of a O-SAML model allows no arguing about a whole O-SAML model. Nevertheless this is not of relevance within this thesis.

To address a certain instance of a O-SAML model, a family parameter can be used. As the shape of the family parameter depends on the O-SAML model, it is not further specified. A obvious way to setup the parameter is to take the value for each param constant and the index of the desired module in each param module group in the model. This leads to a straight forward implementation that allows the creation of arbitrary instances of a given O-SAML model. Any specific instances of an O-SAML model can then be analyzed like a common SAML model.
rule : (metaconstant | formula)* metamodule+ EOF

metaconstant:  constant | paramconst
metamodule:  module | parammodule

paramconst:  param constant vartype BEZ : [ VALUE .. VALUE ];
parammodule:  param module BEZ module+ endparam

formula :  formula BEZ := condition

constant:  constant vartype BEZ ( := VALUE)? ;
module:  module BEZ declaration+ update+ endmodule

declaration :  BEZ : [ INT .. INT ] init INT ;
update:  condition -&gt; assigns ;

assigns:  (probassigns | nondetassigns)
nondetassigns:  nondetassign (+ nondetassign)*
nondetassign:  choice: ( probassigns )
probassigns:  probassign (+ probassign)*
probassign:  (probability : )? nextstassign ( & nextstassign )*nextstassign:  ( BEZ ' = stateexpr )

condition:  conjunction ( | conjunction)*
conjunction:  literal ( & literal)*
literal:  !expression | expression
expression:  statec | BEZ | true | false | ( condition )

statec :  stateexpr ( = | ≤ | < | ≥ | > ) stateexpr

stateexpr : multexpr (( + | − ) multexpr)*
multexpr:  sign2 (( * | / ) sign2)*
sign2:  ( + | negation )* expr
expr:  BEZ | intval | ( stateexpr )

probability :  mult (( + | − ) mult)*
mult :  sign (( * | / ) sign)*
sign:  ( + | − )* prob
prob:  BEZ | VALUE | ( probability )

vartype:  double | float | int

Figure 3.5.: Syntax of the optimizable safety analysis modeling language
4. Artificial Neural Networks

This far the multi-objective optimization problem and its solution with genetic algorithms were introduced (Section 2). In Section 3, O-SAML is introduced as formal specification language with enough degree of freedom for optimization. Thus a genetic algorithm can be applied to optimize a formal specification expressed in O-SAML.

However, the qualitative safety analysis of SAML models requires a high amount of computation time. Even the simple model of an uninterruptible power supply (Section 5) consumes several minutes of computation time on a single core of an eight core Intel Xeon E5430 CPU at 2.66GHz. Taking into account, that genetic algorithms require a large number of objective function evaluations, the optimization of a formal specification is a time consuming task.

An approach to shorten the required computation time of the optimization process is to use an estimation of the objective function. This estimation, for sure, can be done based on meta information of the objective function (of the O-SAML model in this case). Such meta information, however, might not always be available. A more generic estimation approach is to use an estimation algorithm that supports learning by example. The result would be a combination of function estimations, the training of the estimation algorithm and finally the estimation of the objective function.

An artificial neural network is an estimation/approximation algorithm that supports training by example. This section introduces artificial neural networks and shows how they can be integrated as estimation algorithm into the genetic algorithm.

4.1. Idea of Neural Networks

The nervous system is a very complex network of rather simple cells. Each of these cells has several inputs and one output. A single cell is named neuron. The output of the neuron equally depends on all inputs. If an output signal is generated it is said that the cell fires. Whether a cell fires or not, depends on the signals received by the neuron via its inputs. If the input signals exceed a certain threshold, the neuron fires.

It is clear that a simple setup of only one cell cannot perform any complex tasks. However, difficult problems can be solved if several neurons are connected to a neural network. According to estimations, the human brain consists of over $10^{10}$ neurons. Each of these neurons has approximately 7000 connections. In total there
are approximately $1.5 \cdot 10^{14}$ interconnections [19]. These enormous numbers enable the human brain to perform quiet complex tasks.

The overall behavior of the neural network only depends on the interconnections of the neurons and the configuration of each individual cell (i.e. the transfer function of each neuron). The interesting effect of a neuronal network is that, after proper training, it can generalize. This means it can perform operations that were unknown at the design time of the network. In nature neural networks can usually learn on their own. When artificial neural networks come into account the idea of a network of simple processing units is adopted. Nevertheless there are self learning approaches, artificial neural networks often are trained only during the design time. At the run time the network is then used to perform the trained task. The goal of artificial neural networks is to gain a teachable structure with the ability to generalize.

This section gives a brief introduction to the algorithms of artificial neural networks. More information on the topic can be found in the literature [5], [20].

Such incredible high numbers of neurons and connections, as they are found in the nature, are of course not feasibly in a technical system. Nevertheless it is enough to use only a moderately small number of neurons to solve many real world estimation and classification problems.

![Figure 4.1.: Block diagram of a simple artificial neuron with three inputs](image)

### 4.2. Artificial Neural Networks

For an artificial neural network each neuron is represented by a function of the weighted sum of all input signals. This is shown as block diagram in Figure 4.1.
With the inputs to the neuron denoted as $x_i$ and the weight applied to the input $i$ being denoted as $\omega_i$, the output $y(x)$ is calculated as follows:

$$y(x) = G \left( \sum_{i=1}^{n} x_i \cdot \omega_i \right)$$

(4.1)

The function $G$ is called the activation function of the neuron. The activation function evaluates the weighted sum over all input signals of the neuron and decides if the neuron fires. The simplest decision if a neuron fires is a threshold or sign function:

$$G(s) = \begin{cases} 
1 & \text{if } s \geq 0 \\
-1 & \text{if } s < 0
\end{cases}$$

(4.2)

If the input value exceeds a certain threshold (zero in this case) the neuron fires. A disadvantage of this activation function is that it is not differentiable at the threshold point, which is required for some training algorithms. A differentiable approximation of the threshold function is the sigmoid function:

$$G(s) = \frac{1}{1 + e^{-as}}$$

(4.3)

The sigmoid function is steep around the threshold and flat in all other areas. Both, the sign and the sigmoid, activation functions have a bounded output value. However, some applications require an unbounded output of the network. This can be provided by a linear activation function:

$$G(s) = a \cdot s$$

(4.4)

The shape of all three activation functions is illustrated in Figure 4.2.

![Figure 4.2.: The shape of the Sigmoid, Linear and Sign activation functions](image)

4.3. Network Structure

With the neuron as atomic processing unit various different interconnection schemes are thinkable. One of the simplest and widely spread structures is the multilayer
feed forward network. For this type of network the neurons are arranged in layers. The input of a neuron in one layer is connected to all outputs of the neurons of the preceding layer. An example of a multilayer feed forward network with four layers is depicted in Figure 4.3. The depicted network has four inputs that are connected to four neurons in the input layer, which are then followed by two hidden layers. The least layer is the output layer with one neuron and therefore only one output signal.

![Diagram of a multilayer feed forward artificial neural network with two hidden layers](image)

Figure 4.3.: A multilayer feed forward artificial neural network with two hidden layers

In most applications of artificial neural networks the input layer consists of simplified neurons. These only provide the input signal to the internal network and do not have an activation function or weights.

A variant of the multilayer feed forward is if there is only an input and an output layer. It can be shown that a single layer network (all input layer neurons only feed the input signals to the network, thus the input layer is not counted) is sufficient for a classification task where all points can be separated by a hyper plane. A classification that cannot be separated by a hyper plane, like the x-or function, can not be emulated with a single layer network.

The classification according to the logical and and the exclusive or are depicted in Figure 4.4. The separating hyper plane is depicted as dashed line. For the and function only one plane is sufficient. In case of the xor problem, two hyper planes are required.

It can also be shown that a network with one hidden layer is sufficient to approximate any real valued function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ up to any desired precision [5]. This is based on the Kolmogorov theorem:
Theorem 4.3.1 Any continues, real valued, function of the form \( f(x_1, x_2, \ldots, x_n) \) defined on \([0, 1]^n\), \( n \geq 2 \) can be represented in the form

\[
f(x_1, x_2, \ldots, x_n) = \sum_{j=1}^{2n+1} g_j \left( \sum_{i=1}^{n} h_{ij}(x_i) \right)
\]  

(4.5)

where \( g_i \) are properly chosen continuous functions of one variable and \( h_{ij} \) are properly chosen continuous monotonically increasing functions of one variable. The functions \( h_{ij} \) must be independent from \( g_j \).

By comparing this with the Equation (4.1) it is made clear that an artificial neural network with one neuron in the output layer and a single hidden layer can realize Kolmogorov's theorem.

In a generalized version it can be shown that network with sigmoid activation functions in the hidden layer and a linear activation function in the output layer can approximate any real valued function. Due to the linear function in the output layer the network is no longer limited to a closed interval as it is the case for sigmoid functions.

Besides the feed forward structure, other variants exist. These somehow contain a feedback path. As with all systems with a feedback loop their output no longer only depends on the momentary input value \( x \), but also on the history of the input value \( x(t) \) with the time \( t < t_0 \). As a consequence a momentary input value cannot be mapped to a single output value.

Thus network structures with a feedback loop are not feasible for the application as estimators in an optimization process. Therefore the following explanations are limited to multilayer feed forward networks.

4.4. Bias Neuron

The network structure, explained in the previous chapter, has the disadvantage that the activation function of the neurons always is symmetrically around zero.
(compare Equation (4.3)). One solution to this problem would be a parameterized activation function like:

$$G(s) = \frac{1}{1 + e^{-s+a}}$$  \hspace{1cm} (4.6)

However, this parameter must then be determined throughout the training to the network. This would unnecessarily complicate the training algorithm.

Another approach is to add a so called bias neuron to each layer of the network. A bias neuron continuously emits 1 and is connected to all neurons in the subsequent layer. In Figure 4.5 the bias neurons are denoted with small solid black circles.

![Figure 4.5.: Simple network with two bias neurons, depicted as small solid black circles](image)

The bias value is not influenced by any input data, but gets part of the weighted sum on the input of each neuron in the next layer. The weight between the bias neuron and any of the neurons in the next layer controls the influence of the bias neuron on each neuron. By looking at Equation (4.1) and Equation (4.3) it is easy to see that the weighted value of the bias neuron becomes the parameter “a” in equation (4.6).

The obvious advantage of this approach is that the parameter to the activation function can now be trained the same way as every other weight in the network. The training of the network itself is explained in the following chapter.

4.5. Network Training

The application of an artificial neural network is separated in two phases: A training phase and an operation phase. In the training phase the weights in all the neurons are adjusted so that the behavior of the network conforms to the desired behavior. After the network is properly trained it can be used.

One way of training would be to derive the weight parameter directly for the problem that the network should solve. This requires a lot of detailed knowledge about the task that has to be solved and is only feasible for very simple networks with only a few neurons.

A widely used approach is learning by example. Throughout the learning phase a set of examples is presented to the network. An example consists of a vector
of input values $x = [x_1, \ldots, x_i, \ldots, x_I]^T$ for the $I$ inputs and a vector with the expected output value $y = [y_1, \ldots, y_i, \ldots, y_L]^T$. The weights are then modified so that the error between the network output and the expected output is minimized for all training examples.

A famous algorithm for learning by example of artificial neural networks is the Backpropagation learning algorithm. The basic approach of the Backpropagation algorithm is sketched in the next subsection.

Section 4.5 describes an extended version of the Backpropagation learning algorithm.

**Backpropagation Learning**

This is only a brief introduction to the Backpropagation learning algorithm. A more detailed explanation of the algorithm is given in any good literature about artificial neural networks [5], [20].

Backpropagation is a supervised learning algorithm. During the learning phase the network is presented a set of test samples. For each sample the output of the network is compared with the expected output to calculate the error. Based on the error the weights of the output neurons are adjusted in a way so that the error is minimized. For tuning the weights of the hidden-layer neurons, the error (at the output of the network) is propagated backwards through the network towards the input layer. Due to the backward propagation of the error the weights of all layers can be updated to minimize the error at the output of the network.

In one epoch, all the test samples are used one after another to train the network. So for each sample the network weights are updated step by step. Several training epochs are performed to continue the minimization of the output error. The algorithm loops over all training samples until the error falls below a certain threshold or a maximum number of epochs is reached.

In general there are two different versions of the Backpropagation algorithm:

**Batch mode** Update the weights only once in each epoch, after calculating the mean square error over all training samples.

**Incremental** Update the weights after each training sample.

After this informal introduction to Backpropagation learning the mathematical background is sketched. Each iteration of the algorithm consists of two separate operations:

1. Calculate the influence of the weights on the total error of the network.
2. Update the weights so that the total error is minimized.

For convenience the formula symbols used in the following description are illustrated in Figure 4.6. For the description it is assumed that the network has one
output layer and one hidden layer. In each layer only one neuron is depicted. $s_l$ denotes the weighted sum of all inputs to the neuron $l$:

$$s_l = \sum_{j=1}^{L} \omega_{lj} z_j \quad (4.7)$$

$s_j$ is equally defined and denotes the weighted summed in the neuron $j$ on the hidden layer. $y_l$ denotes the output of the neuron on the output layer:

$$y_l = f_o(s_l) = f_o \left( \sum_{j=1}^{L} \omega_{lj} z_j \right) \quad (4.8)$$

And again $z_j$ is defined similarly for the output of the neuron $j$ on the hidden layer. The error function $E$ that shall be minimized during the training process is defined as

$$E = \frac{1}{2} \sum_{l=1}^{L} (d_l - y_l)^2. \quad (4.9)$$

As we will see below the factor $\frac{1}{2}$ is introduced because it leads to a simplification of the derived equations.

Given that the error shall be propagated backwards through the network, from the output layer to the input layer, the weights of the output layer are considered first.

By partially deriving the error function with respect to the weights in the output layer the influence of these weights on the error function can be examined. The chain rule of deviation is applied to decompose the partial deviation:

$$\frac{\partial E}{\partial \omega_{lj}} = \frac{\partial E}{\partial y_l} \frac{\partial y_l}{\partial s_l} \frac{\partial s_l}{\partial \omega_{lj}} \quad (4.10)$$
The first factor results from the partial deviation of Equation (4.9) and leads to the simple term of

\[
\frac{\partial E}{\partial y_l} = -(d_l - y_l). \tag{4.11}
\]

By Equation (4.8) the second factor is the deviation of the activation function of the neuron

\[
\frac{\partial E}{\partial s_l} = f'_o(s_l) \tag{4.12}
\]

and by Equation (4.7) the third factor is simply the input \( j \) to the neuron \( l \):

\[
\frac{\partial s_l}{\partial \omega_{lj}} = z_j. \tag{4.13}
\]

Substituting (4.11), (4.12) and (4.13) in Equation (4.10) leads to:

\[
\frac{\partial E}{\partial \omega_{lj}} = (d_l - y_l) \cdot f'_o(s_l) \cdot z_j. \tag{4.14}
\]

Now that the partial deviations of the error function with respect to the weights in the first layer is calculated, the partial deviations with respect to the hidden layer directly behind the output layer can be evaluated. Thus again the chain rule of deviation is used to decompose the deviation:

\[
\frac{\partial E}{\partial \omega_{ji}} = \frac{\partial E}{\partial z_j} \frac{\partial z_j}{\partial \omega_{ji}}. \tag{4.15}
\]

At this place it is important to notice that \( \omega_{ji} \) is the weight from the neuron \( i \) on the last hidden layer to the neuron \( j \) in the input layer. In Figure 4.6, \( z_j \) is the output of neuron \( j \) and \( s_j \) is the weighted sum of the inputs of neuron \( j \). With \( s_j \) being defined analogue to Equation (4.7) its partial derivation is

\[
\frac{\partial z_j}{\partial \omega_{ji}} = \frac{\partial}{\partial \omega_{ji}} \sum_{l=1}^{L} x_i \omega_{lj} = x_i, \tag{4.16}
\]

and with \( z_j \) being defined analogue to Equation (4.8) its partial derivation is

\[
\frac{\partial z_j}{\partial j} = \frac{\partial}{\partial j} f_o(j) = f'_o(s_j). \tag{4.17}
\]

Substituting \( y_l \) in the error function (4.9) with Equation (4.8) leads to:

\[
E = \frac{1}{2} \sum_{l=1}^{L} (d_l - f_o(s_l))^2
= \frac{1}{2} \sum_{l=1}^{L} \left( d_l - f_o \left( \sum_{l=1}^{L} \omega_{lj} z_j \right) \right)^2 \tag{4.18}
\]
and its partial derivation with respect to $z_i$ is:

$$
\frac{\partial E}{\partial z_i} = \frac{\partial}{\partial z_i} \left( \frac{1}{2} \sum_{l=1}^{L} (d_l - f_0(s_l))^2 \right) = - \sum_{l=1}^{L} (d_l - f_o(s_l)) \frac{\partial f_o(s_l)}{\partial z_j} = - \sum_{l=1}^{L} (d_l - y_l) f'_o(s_l) \omega_{lj}
$$

(4.19)

Substituting Equations (4.16), (4.17) and (4.19) in Equation (4.15) lead to:

$$
\frac{\partial E}{\partial \omega_{ji}} = \sum_{l=1}^{L} ((d_l - y_l) f'_o(s_l) \omega_{lj} f'_h(s_j) x_i).
$$

(4.20)

By comparing the equations (4.14) and Equation (4.20) a formula can be extracted to propagate the error one layer backwards through the network:

$$
d_j - z_j = \sum_{l=1}^{L} ((d_l - y_l) f'_o(s_l) \omega_{lj})
$$

(4.21)

If there is more than one hidden layer in a network, Equation (4.21) needs to be applied recursively to propagate the error layer by layer backwards through the network. In each layer a formula analogue to Equation (4.14) can be used to calculate the partial derivations of the weights in this layer.

In the second step of the Backpropagation algorithm the weights are updated. Each update of a weight only depends on the value of the partial deviation of the error function with respect to the weight that is to be updated. With $\omega_{c}$ being the current weight, the update of every weight $\omega_{\sim}$ is defined as

$$
\Delta \omega_{\sim} = -\rho_0 \frac{\partial E}{\partial \omega_{\sim}}.
$$

(4.22)

The additional factor $\rho_0$ is called the learning rate and is used to balance the influence of all test samples presented during the learning process.

With $\omega_{\sim}^{\text{new}}$ being the new weight value, all weights are finally updated:

$$
\omega_{\sim}^{\text{new}} = \Delta \omega_{\sim} + \omega_{c}.
$$

(4.23)

**Improved Resilient Propagation**

The Backpropagation learning algorithm, described in the last section, is one of the most famous learning algorithms for artificial neural networks. However, the learning rate $\rho_0$ is hard to determine. If it is too small, the learning process continues
very slow and a high number of epochs is needed to properly train the network. If the learning rate is set too high the learning process may start to oscillate and might never reach a stable state.

However, there are approaches to overcome this drawback of the Backpropagation algorithm. One is the resilient propagation algorithm, RPROP [21]. Informally spoken the idea of the RPROP algorithm is to control the learning rate throughout the learning process separately and automatically for each weight. Further on the quantity how much a weight is changed ($\Delta \omega_{(t)}$), no longer depends on the value of the partial derivations of the error function. In contrast the weight update solely depends on the evolution of the leading sign changes of the partial deviation of the error function ($\frac{\partial E}{\partial \omega_{(t)}}$).

With iRPROP there is even an improved version of the RPROP algorithm [22]. It continues the idea to link the weight update to the evolution of the leading sign, but evolves $\Delta \omega_{(t)}$ over the time by a different schema. This section explains the iRPROP algorithm.

During the training process there is an individual weight update $\Delta \omega_{(t)}$ for each weight. The evolution of the weight update is controlled by an adaptive step size, $\Delta_{(t)}$. The evolution of $\Delta_{(t)}$ over the time is controlled by the following rule:

$$
\Delta_{(t)} = \begin{cases} 
\min \left( \eta^+ \cdot \Delta_{(t-1)}, \Delta_{max} \right) & \text{if } \frac{\partial E}{\partial \omega_{(t-1)}} \cdot \frac{\partial E}{\partial \omega_{(t)}} > 0 \\
\max \left( \eta^- \cdot \Delta_{(t-1)}, \Delta_{min} \right) & \text{if } \frac{\partial E}{\partial \omega_{(t-1)}} \cdot \frac{\partial E}{\partial \omega_{(t)}} < 0 \\
\Delta_{(t-1)} & \text{else}
\end{cases} 
$$

(4.24)

with $\eta^- < 1 < \eta^+$. The update factor evolves in a way so that, if the partial derivations of the error function in two consecutive steps possess the same sign, the update factor is increased. If the sign of the partial derivation has changed, the algorithm has jumped over a local minimum of the error function. This means that the training process is close to a local optimum and consequently the step size is decreased.

It is important to mention that the step size is bounded by the parameters $\Delta_{max}$ and $\Delta_{min}$ so that $\Delta_{(t)}$ cannot grow infinitely over time. Obviously, for the algorithm to make sense, these parameters must be chosen so that $0 \leq \Delta_{min} < \Delta_{max}$ holds.

After the step size has been determined, the weight update itself is calculated according to the Algorithm 2.

This again relies on the changing of the sign of the partial derivative. If the sign does not change, the weight update is set so that the weight is moved towards the local optimum. The amount of how much the weight is changed is taken from the step size value. If the sign has changed (a local optimum has passed) and the error has increased, the update step is undone by negating the weight update value from the last time step ($\Delta \omega_{(t-1)}$). If only the sign has changed and the error still decreased, the weight update is not changed. For the last case the weight update will decrease in the next iteration and the search will continue in the opposite direction and thus towards the local optimum.
Algorithm 2 Determine the weight update

\[
\text{if } \frac{\partial E}{\partial \omega}^{(t-1)} \cdot \frac{\partial E}{\partial \omega}^{(t)} > 0 \text{ then }
\]
\[
\Delta \omega^{(t)} := -\text{sign} \left( \frac{\partial E}{\partial \omega}^{(t)} \right) \cdot \Delta \omega^{(t)}
\]
\[
\text{else}
\]
\[
\text{if } E^{(t)} > E^{(t-1)} \text{ then }
\]
\[
\Delta \omega^{(t)} := -\Delta \omega^{(t-1)}
\]
\[
\frac{\partial E}{\partial \omega}^{(t)} := 0
\]
\[
\text{else}
\]
\[
\Delta \omega^{(t)} := \Delta \omega^{(t-1)}
\]
\[
\text{end if}
\]
\[
\text{end if}
\]

Setting the derivative, in the second step of the outer else branch to zero, is an implementation trick. It leads to the else clause in equation (4.24) and thus the learning rate is not updated in the next iteration, so that the modification of the weight is undone.

Finally the weight itself is updated:

\[
\omega^{(t+1)} = \omega^{(t)} + \Delta \omega^{(t)}
\]  

Equation (4.24), Algorithm 2 and Equation (4.25) are evaluated for every weight in every learning iteration and for every learning sample.

4.6. Libfann

Rather than implementing an artificial neuronal network and the learning algorithms from scratch, a preexisting open source library was chosen. With JOONE\(^1\), Neuroph\(^2\), Encog\(^3\) and Libfann\(^4\) [23] there are several projects available.

For this thesis Libfann version 2.1.0beta was chosen. According to its documentation it is heavily optimized for run-time performance. Further on it is a very small and straight forward library. It brings no ballast on top of its core functionality. So basically it supports artificial neural network simulation and a set of training algorithms. To integrate the native C library into Java, fannj\(^5\) version 0.4 was used.

However, for the implementation of this thesis, the neural network has been wrapped in a separate estimation component. Thus it should be easy to adopt the project so that other artificial neural network libraries such as Encog can be used as well. A advantage of Encog could be its GPU support.

4.7. Integrating Artificial Neural Networks in Genetic Algorithms

4.7.1. Estimation Strategy

There are different thinkable variants to integrate an estimator in an evolutionary algorithm.

- Entirely operate the optimization (at least for a time) with the estimated fitness function.

- Use the estimator as a filter prior to the expensive fitness-function evaluation.

The first mentioned strategy is proposed by Nain and Deb [14]. Their estimation strategy is illustrated in Figure 4.7. After an initial phase of $a$ generations the evolutionary algorithm operates for $b - a$ generations with the estimated fitness functions. After that, all estimated individuals in the current generation are evaluated with the true fitness function and the algorithm continue for another $a$ generations with the true fitness function.

Another approach in proposed by Güdemann and Ortmeier [24]. They use the estimator as a filter prior to the expensive fitness-function evaluation. The goal is to prevent the algorithm from wasting time in the evaluation of individuals that are expected to be too weak. This is illustrated in Figure 4.8. After the creation of a new individual (by selection, crossover and mutation) the individuals’ fitness is estimated. Based on this guess the individual is either dropped or passed on to the evaluator.

In this thesis the approach to filter the individuals prior to the evaluation is used. To relieve the selection pressure on the estimator the estimation strategy is slightly modified:

1. Create $n$, $n \geq 2 \cdot PopulationSize$ individuals
2. Estimate all $n$ individuals

3. Apply fast non-dominated sorting to all estimated individuals

4. Drop the weakest half ($\frac{n}{2}$) individuals

5. Randomly choose the desired number of individuals for the next generation out of the remaining $\frac{n}{2}$ individuals.

6. Evaluate all chosen individuals with the true fitness function.

The new created individuals are then combined with the current population and the next generation is extracted according to the NSGA-II algorithm.

To randomly choose the final individuals was motivated by the assumption that this might increase the spread of the Pareto optimal points. Merely choosing the best solutions according to non-dominated sorting might favor areas that are estimated to be outstanding good by the estimator.

### 4.7.2. Shape of the Estimation Network

As an estimation network a multilayer feed forward artificial neural network is used. The network is fully connected. This means, each neuron in one layer is connected
to all neurons in the previous layer.

For a certain estimator all hidden layers have the same number of neurons. The exact number of neurons per hidden layer is a parameter that shall be a whole-number multiple of the number of decision variables. The next parameter is the number of hidden layers. In this thesis one and two hidden layers are examined.

The number of neurons in the output layers depends on the number of objective functions that need estimation. This applies exclusively to the expensive objective functions. Objective functions that are given as simple mathematical expressions on the decision variables shall even be evaluated during the estimation phase.
5. Description of the Case Study

This section describes the case study used to evaluate the optimization approach from this thesis. First the model is explained and second the optimization results of the model are presented.

5.1. Description of the Model

The USV (German abbreviation for: unterbrechungsfreie Stromversorgung) case study features a model of a UPS (uninterruptible power supply). The intention of an UPS is to protect another critical system from power breakdown. Thus the UPS consists of batteries and a diesel engine. A block diagram of the UPS is depicted in Figure 5.1. If the main power fails, the batteries immediately provide power to the system at stake. The batteries only last for a certain time. So if the grid power is back, after a short delay, everything is fine and the batteries can be recharged. If the main power is not restored after a certain time the diesel engine is started. In comparison to the batteries this takes a long time. After the diesel engine is properly started, it can take over the power supply. The diesel engine then drives the system until the main power has been restored. The engine is also strong enough to recharge the batteries. The complete listing of the SAML model is given in Appendix A.

![Block diagram of the uninterruptible power supply.](image-url)

An engineer who is designing the UPS has several decisions to make, which influence the reliability and the costs of the system. Thus the specification of the UPS has the following parameters:

- $n_{\text{batt}}$: The number of the batteries (integer $[0..40]$)
- \( n_{\text{thres}} \): The battery threshold level at which the diesel engine is started (integer \([0..41]\))
- \( \lambda_{\text{batt}} \): The failure probability of the batteries (double \([0..1]\))
- \( d \): Two different types of diesel engines (integer \([0..1]\))

The optimality of the system is distinguished along two objective functions. The first one is the hazard probability:

\[
f_1(n_{\text{batt}}, n_{\text{thres}}, \lambda_{\text{batt}}, d) := P[\text{true} \cup H]
\]

The hazard is defined as the situation where the system at stake is not supplied with energy, neither from the main grid nor from the UPS. This occurs if the grid power fails and the UPS fails to provide the energy.

The second objective function are the system costs. The costs are stated in a way that a decreasing failure probability of a battery leads to an exponential rise of the costs per battery. The number of batteries is multiplied with the price per battery. Further on it is assumed that starting the generator too early also leads to additional costs. Finally there are two fixed cost values for the diesel engine. The complete costs calculate as follows:

\[
f_2(n_{\text{batt}}, n_{\text{thres}}, \lambda_{\text{batt}}, d) := \begin{cases} -\log(\lambda_{\text{batt}}) n_{\text{batt}} + \min(n_{\text{batt}}, n_{\text{thres}}) + 20 & \text{if } d = 0 \\ -\log(\lambda_{\text{batt}}) n_{\text{batt}} + \min(n_{\text{batt}}, n_{\text{thres}}) + 25 & \text{if } d = 1 \end{cases}
\]

The UPS is modeled in a way, that one time step in the model corresponds to 1 minute in real time. The system is examined with a mission time of 24h (equal to 1440 steps of the model).

The SAML model itself is straightforward. The complete listing is presented in Appendix A. The behavior of the two major components (the battery and the diesel engine) are implemented each in its own module. The battery is basically implemented as a counter that decrements while discharging and increments while charging. Each of the maximal 40 batteries lasts for one minute.

The model actually assumes that there are two different types of engines available. Thus they are implemented as a param module group that contains one module for each variant (\(\text{diesel}_0\) and \(\text{diesel}_1\)). The diesel modules consist of a counter which counts the start-up and shutdown time. The difference between the two variants is within the total start-up and shutdown time. The first variant takes nine time steps to start and two to shut down. The second variant is approximately twice as fast. It takes four steps for the start-up and one for the shutdown.

The output switch, depicted in Figure 5.1, is implemented in the module buffer. In the model this module counts the consecutive time steps in which the output is not supplied with power. A real UPS is able to switch to battery supply within a few milliseconds. Based on the basic time unit of the module (one minute) the precise behavior cannot be modeled. It is also not possible to activate the battery
or the generator exactly at the same moment when the mains power fails, because the detection of the power failure takes exactly one time step. To ease this problem the hazard occurs if the counter in the buffer module reaches the state two.

For proper analysis a trivial environment model, that emulates the power grid, is implemented. The environment model emulates the failing of the main grid for a duration from one up to 60 minutes. It is implemented in the module \textit{e\_grid}. The probability that the main power fails is fix set to $1.9 \cdot 10^{-6}$. This is derived from the assumption that the main power fails for one time slice in a whole year. The duration in time slices, of the main power failure is non-deterministic chosen out of these values: $[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60]$

The following failures were identified for the UPS:

- The battery subsystem may fail. This is modeled as a persistent per-time failure. The occurrence probability of this failure is specified by the parameter $\lambda_{\text{batt}}$ and is thus one of the decision variables for the optimization. This failure is implemented in the module \textit{e\_battery}.

- The diesel engine may fail to start. This is modeled as persistent on-demand failure. This failure is modeled with an occurrence probability of $0.0001$ (one out of 10000 start procedures fail). This failure is implemented in the module \textit{e\_diesel\_start}. As a per-demand failure this module can only change into the fail state if the diesel engine is requested to start. The start request is covered in the formula \textit{diesel\_request}.

- The diesel engine may fail to operate. This is modeled as a persistent per-time failure. The occurrence probability of this failure is derived from the assumption that the diesel may fail once a year and is modeled as $1.9 \cdot 10^{-7}$. This failure is implemented in the module \textit{e\_diesel}.

A disadvantage of this model is that there are several invalid model parameter settings. This applies if the diesel start threshold is greater than the maximum number of batteries available. To compensate the influence of the illegal parameter combinations, an additional filter was added to the optimizer. This filter only allows the generation of valid solutions. The filter applies to the crossover and mutation operator. These operators are repeated for each offspring individual that is to be created, until a valid child is created.

5.2. Analysis of the Model

The USV model was optimized with the NSGA-II algorithm, with and without estimator. For the Pareto front the results of all evaluations from several separate optimization runs were combined\(^1\). Each of the runs had 25 generations with 25 individuals each. In each run an individual population of 25 individuals was created.

\(^1\)The NSGA-II algorithm considers the first front of the last population of one single run as the optimal set. For this optimization each individual was also stored in one external storage
The Pareto front is depicted in Figure 5.2. The Figures 5.3 to 5.8 depict the relationship between the two objective functions and each of the parameters: number of batteries, diesel start threshold, battery failure probability and diesel engine type.

Figure 5.2.: The Pareto front of the USV model with logarithmic scaling on both axis.

for each run. At the end the Pareto front was extracted from all individuals in the external storage. All fronts from each run were then united and the final front was extracted.
Figure 5.3.: Both objectives of the USV plotted over the number of batteries.

Figure 5.4.: Hazard probability of the USV plotted over the diesel start threshold
Figure 5.5.: System costs of the USV plotted over the diesel start threshold

Figure 5.6.: Both objectives of the USV plotted over the battery failure probability
Figure 5.7.: Hazard probability of the USV plotted over the generator type

Figure 5.8.: System costs of the USV plotted over the generator type
6. Implementation

This chapter covers the implementation of the software that was developed within this thesis. First of all the overall software architecture is introduced. After that all important components of the system are described briefly.

6.1. Overall Software Architecture

For the whole system two separate tasks were identified. On the one hand there is the optimizer related system and on the other hand there are the objectives. One of the main objectives in this thesis is the SAML based qualitative safety analysis.

![Diagram](image)

Figure 6.1.: The connection between the optimization related and SAML related code

The relation between the two domains is illustrated in Figure 6.1. The interface IObjectiveFunctions covers the functionality of the objective function vector. On the left side the NSGA2 optimizer uses this interface as problem statement. The SAML related classes are on the right side. These consist of the SamlSystemCluster class, that encapsulates a complete O-SAML model. The SamlSystemClusterParam covers the instantiation parameter for one specific O-SAML instance.

6.2. SAML/O-SAML Related Software

The O-SAML related code relies on the pre-existing implementation of the SAML parser and transformation routines. The SAML related code (parser, transformations into different output languages) is not covered in this thesis. An early prototype of the O-SAML approach [25] was also used for the implementation.
However, the O-SAML related code was partially refactored and modified to fit the specific requirements of the optimization procedure used in this thesis and the SAML parser was extended to read O-SAML models.

![Class diagram of the SAML related code](image)

**Figure 6.2.:** Class diagram of the SAML related code

The detailed structure of the objective functions is illustrated in Figure 6.2. The *PrismObjective* is a generic objective that uses the PRISM model checker to perform a quantitative analysis of a SAML model (*SamlSystem*). A *SamlSystem* is an instance of an O-SAML model, i.e. a *SamlSystemCluster*. The *UsvCostObjective* is implemented especially for the case study of the uninterruptible power supply. It uses the O-SAML instance parameters (*SamlSystemClusterParam*) to evaluate the system costs. All objectives are registered to the *SamlSystemCluster*. This is indicated by the association in Figure 6.2.

It is important to mention that the *IObjectiveFunctions* interface in Figure 6.1 is not related to the *IObje ctive* interface in Figure 6.2. The first one is used by the optimizer and covers a whole objective function vector. The second one covers the functionality of one certain objective that is related to an O-SAML model. This enables good reusability of the code. On one side new O-SAML related objectives are easy to implement. On the other side the optimization algorithm can be reused as a standalone component for a completely different optimization task. The *SamlObjectiveFunctions* class (which implements the *IObjectiveFunctions* interface) is implemented so that it can deal with arbitrary sets of emph*IObjective* instances.

### 6.3. Optimizer Related Software

Although most of the optimization related code was developed solely for this thesis, some minor parts were adopted from the NSGA-II reference implementation\(^1\) and

\(^{13}\) January 2012: Version 1.1.5 from http://www.iitk.ac.in/kangal/codes.shtml
Figure 6.3.: Class diagram of the optimizer related code
the jMetal\textsuperscript{2} project. The affected regions are properly labeled in the source code.

The structure of the optimizer is depicted in Figure 6.3. On the right side the relation between the \texttt{NSGA2} class and the \texttt{SamlObjectiveFunctions} class, known from Figure 6.1, is shown. The optimizer internally uses the \texttt{VariableSet} as a representation of the individuals. The \texttt{VariableSet} contains two lists of \texttt{IVariables}. One list represents the chromosome and the other represents the genotype.

In case of the NSGA-II implementation the \texttt{VariableSet} is wrapped in the \texttt{DominanceCrowdingUnit}. This class holds the description of an individual (as \texttt{VariableSet}), the Pareto rank and the crowding distance value.

The \texttt{DominanceCrowdingUnit} implements the \texttt{IDominanceCrowdingUnit} interface, which itself contains the interfaces \texttt{ICrowdingUnit} and \texttt{IDominanceUnit}. The first one provides an interface to the get and set the crowding distance that is assigned to an individual. The second one provides an interface used by the fast non-dominated sorting algorithm.

The class \texttt{CrowdingDistanceHelper} is used to assign the crowding distance. The \texttt{CrowdingComparisonOperator} implements the crowded comparison operator that is used in the NSGA-II algorithm. This operator requires the crowding distance and the Pareto rank. Thus it uses the \texttt{IDominanceCrowdingUnit} interface. The class \texttt{FastNondominatedSort} implements the fast non-dominated sorting algorithm. The class \texttt{SBXCrossover} implements the simulated binary crossover operator.

All operators mentioned in the last paragraph are used by the NSGA-II implementation to perform the certain tasks. This again allows good reusability of all components in different contexts. A different optimization algorithm could for example use the fast non-dominated sorting but a different schema to calculate the crowding distance.

The implementation of the NSGA-II algorithm also uses the \texttt{IEstimator} interface. Thus it is not a strict implementation of the original NSGA-II algorithm, but covers certain extensions and modification that are proposes throughout this thesis.

The estimator part is illustrated in Figure 6.4. In this case only one imple-


![Figure 6.4.: Class diagram of the estimator related code](image)

62
Figure 6.5.: The numerical data types used by the optimizer

The implementation of the IEstimator interface exists. This is the artificial neural network estimator, which is based on libfann.

The available realisations of the IVariable interface are illustrated in Figure 6.5. The class Real covers a real number. The class Number represents an integer number. According to the common interface, both values support an upper and a lower bond. The interface grants access to the values in double precision floating point representation. The class Number additionally offers access to the values in the more appropriate long integer representation.

The separation in real and integer valued representations enables the usage of different operators (crossover and mutation) according to the required encoding. This feature, however, was not used in this thesis, but it might become handy in some future extensions.
7. Evaluation

7.1. Analysis Methods

This section covers the analysis methods used to compare the various optimization experiments. The first difficulty is that the Pareto front of none of the optimization functions is known, nor can it be calculated easily. Thus the reference front was taken from several NSGA-II based optimization processes. However, the resulting front remains a set of points. For the analysis the front was interpolated. The interpolation was done so that the worse case between all points is assumed. This is illustrated in Figure 7.1.

![Figure 7.1.: Worse case interpolation of a Pareto front.](image)

This interpolation scheme uses the natural border between all non-dominated points and all points that are dominated by the reference set. Based on this interpolation the distance between any point and the reference set is calculated according to the algorithm 7.1.

In the first step, the set of points is extended by one additional point for each objective. This point is chosen in a way, so that the interpolation does not end at the outermost points.

After that the reference set is sorted by each objective one after another. After sorting by one objective the index, after which the desired point \( I \) fits in, is deter-
Algorithm 3 Distance calculation between a point and the Pareto front.

\begin{algorithm}
\begin{algorithmic}
\Function{distanceCalculation}{point, ParetoFront}
\For{$m = 0$ to $\dim(PF^*)$}
\State $A = [-\infty, \ldots, -\infty]$
\State $A_m = \infty$
\State $PF^* = PF^* \cup A$
\EndFor
\For{$m = 0$ to $\dim(PF^*)$}
\State $PF^* = PF^* \cup I$
\State $Q = \text{sort}(PF^*, m)$
\State $i = \text{index}(I, PF^*, m)$
\For{each $j$ in $\{1, \ldots, \dim(PF^*)\} \setminus m$}
\State $tmp_j = |I_j - PF^*_{i,j}|$
\EndFor
\State $D_m = \min(tmp)$
\EndFor
\State $dist = \min(D_m)$
\If{$point \preceq PF^*$}
\State $dist = -1 \cdot dist$
\EndIf
\EndFunction
\end{algorithmic}
\end{algorithm}

mined by the index function. Then distance along all objectives, except the one used for sorting, is calculated and the minimal distance is stored in a vector.

After the sorting and distance calculation along all objectives, the resulting distance is the minimum out of all previously calculated distance values.

Finally the sign of the distance value is corrected. A Positive distance value is assumed for any point that is dominated by the reference front. A negative distance value indicates a point that is not dominated by the reference set.

The distance calculation is illustrated for two objective functions in Figure 7.2. In the illustration the resulting distance is the minimum of $d_1$ and $d_2$. The dashed lines illustrate the purpose of sorting and index determination in the distance algorithm.

Although the algorithm is defined for an arbitrary number of objectives ($> 2$), it is used with two objectives only. However, two objectives were enough for the case study used in this thesis.

7.2. Reference Front

The reference front was taken from 56 optimizations with each 650 function evaluations. The results from all runs were combined, and the non-dominated solutions were extracted. This leads to a reference front with 1400 points.

Due to the observation, that the two objectives are spread in different ranges, the second objective was rescaled. The rescaling was done by a factor of $\frac{1}{15000}$. Thus the second objective falls into the same co-domain as the first objective.
After the rescaling, the reference front was shifted for an absolute value of 0.1 on both objectives towards the optimality. Thereby an unreachable reference front was created.

With \([f_1, f_2]^T\) denoting the two objectives, the proper rescaling was done by

\[
\begin{bmatrix}
  f_{1r} \\
  f_{2r}
\end{bmatrix} = \left( \begin{bmatrix}
  1 & 0 \\
  0 & \frac{1}{15000}
\end{bmatrix} \cdot \begin{bmatrix}
  f_1 \\
  f_2
\end{bmatrix} \right) - \begin{bmatrix}
  0.1 \\
  0.1
\end{bmatrix}.
\] (7.1)

Of course the same rescaling (without the shift) was also done on the results that were to compare with the reference front. Thus the expected distance between the experiment results and the reference front is approximately 0.1.

7.3. Results

All in all three different approaches to determine and evaluate the influence of the estimator were tested. The next three sections cover all three approaches.

7.3.1. Number of Dominated Points

For the first evaluation approach the results of optimization runs with the estimator were compared with the results of runs without estimator. The comparison was made by counting how many points in a run with estimator dominate the front that was generated without optimizer. For this evaluation the reference front was not rescaled and not shifted. Otherwise the domination criterion would have been senseless.
However, the plain binary comparison did not show any difference between different estimator configurations. It even did not show any differences between runs with and without an estimator. Thus this evaluation approach was dropped very early.

7.3.2. Convergence Speed

This experiment tried to determine the influence of the estimator on the convergence speed of the algorithm towards the reference front. The idea is that an optimizer with an estimator should converge faster than the one without estimator.

<table>
<thead>
<tr>
<th>Name</th>
<th>Generations</th>
<th>Pop. Size</th>
<th>Nr. of Hidden Layer</th>
<th>Neurons/Hid. Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSGA</td>
<td>25</td>
<td>25</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ANN-1-8</td>
<td>25</td>
<td>25</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>ANN-2-8</td>
<td>25</td>
<td>25</td>
<td>2</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 7.1.: Configuration of the convergence speed experiment

For this analysis three different experiments were performed. Each experiment consisted of 28 equal runs. The configuration of all three experiments are listed in Table 7.1. The configuration of the evolutionary algorithm was not changed in all three experiments. Only the estimator configuration was changed.

Figure 7.3.: Graphical representation of the results of the convergence analysis
After each generation the dominant individuals from all previous generations where extracted and saved. After that the mean distance between all individuals of a generation and the reference front was calculated. Finally the average of all mean distances for each generation was calculated.

The result is plotted in Figure 7.3. On the x-axis the generations are plotted. The y-axis shows the average of all mean distances in this generation.

A clear weakness of this analysis is the fact that the minimal and maximal mean distance in every generation spread very wide. Figure 7.4 shows the data from Figure 7.3 together with the minimal and maximal distance in every generation. It is clear to see, that the difference between the best and the worst run is far more than the overall convergence of the algorithm. Thus this analysis did not show many results. However the single layer network seems to perform less bad results than the double layer artificial neural network.

![Graphical representation of the convergence analysis with error bars](image)

**Figure 7.4.:** Graphical representation of the convergence analysis with error bars

### 7.3.3. Influence of the Population-Size

This experiment tried to determine the influence of the population size on the convergence of the genetic algorithm. Thereto 12 different optimization settings were chosen according to Table 7.2.

The first four experiments were performed without any estimator. For the second four experiments an estimator with one hidden layers and eight neurons per hidden layer was used. The neurons in the output layer had a linear activation function.
The estimator operated on the logarithm of the probability value. For the last four experiments the estimator had two hidden layers.

<table>
<thead>
<tr>
<th>Name</th>
<th>Generations</th>
<th>Population Size</th>
<th>Total Func. Eval.</th>
<th>Estimator</th>
</tr>
</thead>
<tbody>
<tr>
<td>nsga-10</td>
<td>10</td>
<td>64</td>
<td>640</td>
<td>no</td>
</tr>
<tr>
<td>nsga-20</td>
<td>20</td>
<td>32</td>
<td>640</td>
<td>no</td>
</tr>
<tr>
<td>nsga-30</td>
<td>30</td>
<td>21</td>
<td>630</td>
<td>no</td>
</tr>
<tr>
<td>nsga-40</td>
<td>40</td>
<td>16</td>
<td>640</td>
<td>no</td>
</tr>
<tr>
<td>ann-1-10</td>
<td>10</td>
<td>64</td>
<td>640</td>
<td>1 Hidden Layer</td>
</tr>
<tr>
<td>ann-1-20</td>
<td>20</td>
<td>32</td>
<td>640</td>
<td>1 Hidden Layer</td>
</tr>
<tr>
<td>ann-1-30</td>
<td>30</td>
<td>21</td>
<td>630</td>
<td>1 Hidden Layer</td>
</tr>
<tr>
<td>ann-1-40</td>
<td>40</td>
<td>16</td>
<td>640</td>
<td>1 Hidden Layer</td>
</tr>
<tr>
<td>ann-2-10</td>
<td>10</td>
<td>64</td>
<td>640</td>
<td>2 Hidden Layer</td>
</tr>
<tr>
<td>ann-2-20</td>
<td>20</td>
<td>32</td>
<td>640</td>
<td>2 Hidden Layer</td>
</tr>
<tr>
<td>ann-2-30</td>
<td>30</td>
<td>21</td>
<td>630</td>
<td>2 Hidden Layer</td>
</tr>
<tr>
<td>ann-2-40</td>
<td>40</td>
<td>16</td>
<td>640</td>
<td>2 Hidden Layer</td>
</tr>
</tbody>
</table>

Table 7.2.: Configuration of the experiments with variable population size.

For all experiments the number of generations and the population size was chosen that overall approximately 640 function evaluations are done during a single optimization. For all experiments a mutation probability of 0.25 and a crossover probability of 0.9 were chosen.

With each configuration, 21 equal repetitions were performed. For each configuration the average of the mean distances over all 21 equal runs was calculated. The results are shown in Table 7.3. Besides the average value the minimum and the maximum out of the 21 repetitions are also listed.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Average Distance</th>
<th>Min. Distance</th>
<th>Max. Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>nsga-10</td>
<td>0.1000076905260375</td>
<td>0.0999999126859011</td>
<td>0.1003826087420640</td>
</tr>
<tr>
<td>nsga-20</td>
<td>0.100055132506573</td>
<td>0.0996691842805428</td>
<td>0.1002984025289223</td>
</tr>
<tr>
<td>nsga-30</td>
<td>0.100064023554882</td>
<td>0.0999995850906781</td>
<td>0.1002337321619562</td>
</tr>
<tr>
<td>nsga-40</td>
<td>0.100145308950969</td>
<td>0.0999998295484311</td>
<td>0.1006931386119068</td>
</tr>
<tr>
<td>ann-1-10</td>
<td>0.100048847094176</td>
<td>0.1000027150969308</td>
<td>0.1001898407737630</td>
</tr>
<tr>
<td>ann-1-20</td>
<td>0.1000278926984611</td>
<td>0.1000010296284272</td>
<td>0.1008540578437402</td>
</tr>
<tr>
<td>ann-1-30</td>
<td>0.100145308950969</td>
<td>0.0999998295484311</td>
<td>0.1005937568359835</td>
</tr>
<tr>
<td>ann-1-40</td>
<td>0.100089491518541</td>
<td>0.0998967726681884</td>
<td>0.1003848886978770</td>
</tr>
</tbody>
</table>

Table 7.3.: Comparison of different population-size settings with and without estimator.
The results listed in Table 7.3 are plotted in Figure 7.5. In this figure the numbers 1...4 on the x-axis correspond to one tenth of the number of generations.

As the number of function evaluations is approximately constant in all experiments, the optimization benefits from the estimator in extreme cases. These are either many generations with a very small population size (nsga-40 and ann-1-40), or few generations with a higher population size (nsga-10 and ann-1-10). The aspect, that the estimator is contra productive in between the both extreme situations (nsga-20, nsga-30, ann-1-20 and ann-1-30) and even in some cases for a small population size (ann-1-10) is critical.

According to the spread within each experiment the single hidden layer estimator shows a good performance. In the experiments ann-1-10 and ann1-20 shows the smallest spread compared to all other experiments. This is a good result as the estimator at least does not slow down the optimization process.

Also notable is the fact, that the spread between the best and the worst run out of the 21 repetitions of one configuration is spread wider if the estimator is used, than without the estimator. Taking this into account, only a rather large generation count of 40 with a smaller population size lead to truly better results than without estimator.

![Figure 7.5: Graphical representation of Table 7.3](image-url)
8. Conclusion

8.1. Related Approaches

A completely different approach is implemented in the PARAM model checker [4]. The initial idea is to directly analyze parametric probabilistic transition systems. The result of such an analysis is not a specific value, but a formula. The resulting formula covers the analysis results for all model variants covered by the parameterized model. This formula is to be evaluated with the desired parameter values to gain the analysis for a certain model instance.

The core of the PARAM model checker is designed to analyze discrete parametric Markov chains [26]. Thus no non deterministic behavior is taken into account. Nevertheless, since version 1.0 the PARAM model checker supports an extension for models with non deterministic behavior (in terms of Markov decision processes). The non determinism in the model is thereto treated as an additional parameter. However, the approach is limited in two aspects. First, only maximum reachability properties are covered. Second, and more important, the costs of the approach scales exponentially with the number of non deterministic choices. This might render the approach useless for models with a complex non deterministic behavior.

It will need further research to determine whether the PARAM model checker is able to deal with similar complex (or even more complex) models. It then could be combined with the evolutionary optimization approach. Thereto the parametric transition system is turned into a formula (which is expected to be cheaper to evaluate). This formula is then used in the optimization process instead of the very expensive PRISM evaluations.

8.2. Suggestions for Future Research

The case studies used in this thesis and in the initial publication [24], both introduce a real valued parameter. This parameter directly influences the modeled failure probability of a certain component of the analyzed system. The fact that a real valued parameter was chosen, increases the difficulty of solving the remaining multi-objective optimization problem. Even though genetic algorithms are able to solve such problems, there are other approaches.

Besides a strict mathematical analysis of the problem, it might be valuable to take additional meta information into account. Such meta information might be, that a strict real valued parameter (bounded or unbounded) is not very realistic in the technical domain:
1. It is rather unlikely that the failure probability of a component can be influenced directly. It is more likely that there is a certain number of interchangeable components that each have a certain failure probability. In O-SAML, this can be expressed in terms of parameter modules. Thus the real valued parameter leads to a discrete parameter.

2. Even if a real valued parameter can be chosen freely, it still can only be selected up to a certain accuracy. A very intuitive example is the speed limitation in public traffic. Consider a speed limit of 120.05 km/h. It is intuitive to see, that the fraction of $\frac{5}{100}$ makes no sense, since it cannot be measured by an odometer that is commonly installed in a car. It is common to treat technical parameters with a relative error. Considering an error of 10% even 120 km/h and 125 km/h make no difference, since the allowed tolerance around both values is greater than the difference between both. This again leads to a discrete valued parameter. A common example from the electrical engineering domain are the E-Series (IEC 60063) that specifies preferred norm values for several electrical components like resistors and capacitors. All values in the E-Series are chosen, so that the tolerance ranges of consecutive values do not overlap.

3. According to the technical meaning of the parameters certain very large or very small parameter values may be senseless. In case of the failure probability parameter, a very high failure probability is unrealistic, since one would not include a component that is expected to fail. Such a component might turn out to be useless, and should be removed from the design. Very low failure probabilities are unrealistic, as it is not possible to design systems components with an arbitrarily low failure probability. This leads to more or less strict bindings of the parameters.

A combination of all three points leads to a drastically reduction of the search space. This could then be used to further assist the evolutionary algorithm.

It will need further research to determine the effort for the optimization of a real world problem under consideration of the restrictions mentioned above. Instead of using brute force or an evolutionary algorithm, it could also be feasible to evaluate the expensive fitness function at certain well chosen points and interpolate the function in between.

Another open topic for further research covers the treatment of possible invalid parameter combinations. These for example appear in the uninterruptible power supply. All configurations where the diesel start threshold is below the maximum number of available batteries are meaningless. In this thesis a specific filter was used to exclude all invariant solutions. One consequent approach would be a generic interface to express constraints on all parameter. However, this requires further investigation.

Last but not least, the O-SAML grammar might require further extensions. It might be useful to allow param modules with different ranges for the state variables.
This would allow easier modeling of certain components such as the different diesel engines in the case study. Both variants have a different number of reachable states but the same state variable configuration must be used. However, if such variation is allowed it becomes a more complicated task to distinguish valid O-SAML models from invalid ones. This implies the question whether the variable state space (caused by the parametric state variable declaration) of a param module is always compatible with the activation conditions in all other modules or param modules.

8.3. Summary

The work presented in this thesis, provides an approach for the optimization of formal specifications. Possible optimization goals are cost, performance or especially safety aspects. The safety aspects are covered by a quantitative model based analysis that is performed by the PRISM model checker.

As each single optimization objective leads to trivial solutions, the task must be treated like a multi-objective optimization problem. This means several antagonistic objectives must be optimized at the same time. Due to the fact, that a multi-objective optimization problem has no single optimal solution, the Pareto dominance criterion is used to compare multiple solutions according to their optimality. As a result, several solutions are calculated in a way, so that none of the solutions dominates any of the other results. The set of optimal solutions is referred to as Pareto front.

A genetic algorithm was applied to solve the multi-objective optimization problem. Genetic algorithms do not require any meta information about the function that needs optimization. This meets the requirements stated by the safety relevant objectives. These objectives often contain a quantitative model based analysis, which itself is a very complex mathematical algorithm. Thus the objective needs to be treated as black-box function with no additional meta information (like derivatives or extrema).

However, even if genetic algorithms are able to solve generic multi-objective optimization problems they rely on a huge number of objective function evaluations. This is not a problem in case of simple mathematical expressions. Yet the quantitative model based safety analysis requires a huge amount of computation time. Thus an artificial neural network is used to estimate the safety analysis and thereby speed up the optimization process.

The safety analysis modeling language is used as formal specification language. SAML itself covers invariant specifications. Thus an extension to SAML is proposed which enables the expression of variant specifications. The variability is provided in terms of variable constants (similar to pre-processor directives in C) and alternative modules. The semantic of an O-SAML model is defined as a family of SAML models. Thus an O-SAML model can be turned into a SAML model by instantiating the O-SAML model. Thereto each param constant is replaced with a constant and
a single module is selected out of every param module group.

The complete algorithms discussed in this thesis were implemented or preexisting libraries were properly included. The implementation was then evaluated on a case study and the ability of the approach was shown.

Due to a flexible software design the implementation of the optimization algorithm is not restricted to the optimization of SAML based formal specifications. Several interfaces are used to modularize the implementation. Even further extensions to different optimization or estimation algorithms are expected to be easy to implement into the existing framework.

The modular implementation enables the implementation of the extensions and alternative approaches proposed in Section 8.1 and Section 8.2 about related work and further research topics. This implies both proper discretisation of real valued parameters and thus reduction of the search space and the inclusion of different analysis methods like the PARM model checker.
A. USV Model

// Model of an uninterruptible powersupply featuring a battery and
// a diesel engine.

// Timebase: 1 step = 1 minute
// Mission time: 24h = 1440 steps

// Hazard: buffer_state = 2
// meaning that there is no power for more than 1 time step

// PCTL: Pmax=?[ true U<=1440 buffer_state = 2 ]

formula no_grid := grid_state > 0;

formula diesel_power := diesel_state = 10;

formula diesel_request := battery_state < diesel_start_level &
 !(battery_state = -1) & no_grid;

formula diesel_request := battery_state < diesel_start_level;

formula diesel_off := diesel_state = 0;

formula battery_power := battery_state > 0;

formula ns_power := no_grid & !diesel_power & !battery_power;

formula grid_fail := e_grid_state = 1;

formula battery_fail := e_battery_state = 1;

formula diesel_start_fail := e_diesel_start_state = 1;

formula diesel_fail := e_diesel_state = 1;

constant double p_grid := 1.9e-6; // 1 timeslice per year
constant double p_diesel_1 := 1.9e-7; // once in 10 year
constant double p_diesel_start_1 := 0.0001; // every 10000 time

//constant int battery_capacity := 30;
// Capacity of the battery 0..40 minutes

param constant int battery_capacity : [0..40];

//constant int diesel_start_level := 10;
// The battery level at which the diesel engine is started:

param constant int diesel_start_level : [0..41];

//constant double p_battery := 0.001;
// The probability that the battery module fails

param constant double p_battery : [0..1];

// Per time failure (healing)
// not when battery in init state!!

module e_grid

e_grid_state : [0..1] init 0;

e_grid_state = 0 & battery_state = -1 ->
choice(1: (e_grid_state’ = 0));

e_grid_state = 0 & !(battery_state = -1) ->
choice(p_grid:(e_grid_state’ = 1) +
1-p_grid: (e_grid_state’=0));

e_grid_state = 1 -> choice(1: (e_grid_state’ = 0));
endmodule

// Per time failure (stuck)

module e_battery

//
```haskell
module battery
  // -2 error state
  // -1 initial state: used to initialize the battery capacity
  // 0 battery empty
  // >0 battery charge level
battery_state : [-2..40] init -1;
  // init
battery_state = -1 & !battery_fail ->
choice(1: (battery_state' = battery_capacity));
  // discharge
battery_state > 0 & !no_grid & !diesel_power & !battery_fail ->
choice(1: (battery_state' = battery_state - 1));
battery_state = 0 & !no_grid & !diesel_power & !battery_fail ->
choice(1: (battery_state' = 0));
  // charge
battery_state < battery_capacity & (!no_grid | diesel_power) &
```

```haskell
module e_batt
  e_batt_state : [0..1] init 0;
e_batt_state = 0 -> choice(p_batt: (e_batt_state' = 1) +
                         1-p_batt: (e_batt_state' = 0));
e_batt_state = 1 -> choice(1: (e_batt_state' = e_batt_state));
```
battery_fail -> choice:(1: (battery_state' = battery_state + 1));
battery_state = battery_capacity & (!no_grid | diesel_power) &
!battery_fail -> choice:(1: (battery_state' = battery_state));
battery_fail -> choice:(1: (battery_state' = -2));
// failure
battery_state = -2 -> choice:(1: (battery_state' = battery_state));
endmodule

param module diesel
module diesel_0
  // -3 error state
  // -2...1 diesel is shutting down
  // 0 diesel is off
  // 1...9 diesel is starting up
  // 10 diesel is running
diesel_state : [-3..10] init 0;
  // diesel is off
diesel_state = 0 & !diesel_request -> choice:(1: (diesel_state' = 0));
  // diesel is off but there is a request no startup failure
diesel_state = 0 & diesel_request & !diesel_start_fail ->
    choice:(1: (diesel_state' = 1));
  // diesel is in startup phase no startup failure
diesel_state > 0 & diesel_state < 10 & !diesel_start_fail ->
    choice:(1: (diesel_state' = diesel_state + 1));
  // diesel is off but there is a request with(!) startup failure
diesel_state = 0 & diesel_request & diesel_start_fail ->
    choice:(1: (diesel_state' = -3));
  // diesel is in startup phase with (!) startup failure
diesel_state > 0 & diesel_state < 10 & diesel_start_fail ->
    choice:(1: (diesel_state' = -3));
  // diesel is running, no grid, no failure: keep running
diesel_state = 10 & no_grid & !diesel_fail ->
    choice:(1: (diesel_state' = diesel_state));
  // diesel is running but grid becomes available
  // and no failure: turn diesel off
diesel_state = 10 & !no_grid & !diesel_fail ->
    choice:(1: (diesel_state' = -2));
  // diesel is running, no grid and failure: goto failure state
diesel_state = 10 & no_grid & diesel_fail ->
    choice:(1: (diesel_state' = -3));
  // diesel is running but grid becomes available and
  // failure: goto failure state
diesel_state = 10 & !no_grid & diesel_fail ->
    choice:(1: (diesel_state' = -3));
  // diesel is shutting down
diesel_state < 0 & diesel_state > -3 ->
    choice:(1: (diesel_state' = diesel_state + 1));
  // failure
diesel_state = -3 -> choice:(1: (diesel_state' = diesel_state));
endmodule

// diesel_1 starts and stops twice as fast as diesel_0
module diesel_1
  // -3 error state
  // -2...1 diesel is shutting down; here only -1 is used
  // 0 diesel is off
  // 1...9 diesel is starting up; here only 2,4,6,8 are used
  // 10 diesel is running
diesel_state : [-3..10] init 0;
  // diesel is off

diesel_state = 0 & !diesel_request -> choice:(1: (diesel_state' = 0));
// diesel is off but there is a request no startup failure
  diesel_state = 0 & diesel_request & !diesel_start_fail ->
    choice:(1: (diesel_state' = 2));
// diesel is in startup phase no startup failure
  diesel_state > 0 & diesel_state < 9 & !diesel_start_fail ->
    choice:(1: (diesel_state' = diesel_state + 2));
// diesel is off but there is a request with(!) startup failure
  diesel_state = 0 & diesel_request & diesel_start_fail ->
    choice:(1: (diesel_state' = -3));
// diesel is in startup phase with (!) startup failure
  diesel_state > 0 & diesel_state < 10 & diesel_start_fail ->
    choice:(1: (diesel_state' = -3));
// diesel is running, no grid, no failure: keep running
  diesel_state = 10 & no_grid & !diesel_fail ->
    choice:(1: (diesel_state' = diesel_state));
// diesel is running but grid becomes available and no
// failure: turn diesel off
  diesel_state = 10 & no_grid & !diesel_fail ->
    choice:(1: (diesel_state' = -1));
// diesel is running, no grid and failure: goto failure state
  diesel_state = 10 & no_grid & diesel_fail ->
    choice:(1: (diesel_state' = -3));
// diesel is running but grid becomes available and
// failure: goto failure state
  diesel_state = 10 & no_grid & diesel_fail ->
    choice:(1: (diesel_state' = -3));
// diesel is shutting down
  diesel_state < 0 & diesel_state > -3 ->
    choice:(1: (diesel_state' = diesel_state + 1));

endmodule
module buffer
  buffer_state : [0..2] init 0;

  buffer_state = 0 & !no_power -> choice:(1: (buffer_state' = 0));
  buffer_state = 0 & no_power -> choice:(1: (buffer_state' = 1));
  buffer_state = 1 & no_power -> choice:(1: (buffer_state' = 2));
  buffer_state = 1 & !no_power -> choice:(1: (buffer_state' = 1));
  buffer_state = 2 & !no_power -> choice:(1: (buffer_state' = 2));
endmodule
Bibliography


Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur mit erlaubten Hilfsmitteln angefertigt habe.

Magdeburg, den January 17, 2012

Simon Struck