# Efficient Optimization of Large Probabilistic Models

Simon Struck[a], Matthias Güdemann[b], Frank Ortmeier[a]

[a] *AG Computer Systems in Engineering, Otto-von-Guericke University Magdeburg, Germany*
[b] *CONVECS, Inria Rhône-Alpes, Grenoble, France*

**Abstract**

The development of safety critical systems often requires design decisions which influence not only dependability, but also other properties which are often even antagonistic to dependability, e.g., cost. Finding good compromises considering different goals while at the same time guaranteeing sufficiently high safety of a system is a very difficult task.

We propose an integrated approach for modeling, analysis and optimization of safety critical systems. It is fully automated with an implementation based on the Eclipse platform. The approach is tool-independent, different analysis tools can be used and there exists an API for the integration of different optimization and estimation algorithms. For safety critical systems, a very important criterion is the hazard occurrence probability, whose computation can be quite costly. Therefore we also provide means to speed up optimization by devising different combinations of stochastic estimators and illustrate how they can be integrated into the approach.

We illustrate the approach on relevant case-studies and provide experimental details to validate its effectiveness and applicability.

**Keywords: safety analysis, formal methods, multi-objective optimization, safety optimization**

## 1. Introduction

In many domains, software has become a major innovation factor. Very often different systems use the same standard hardware, their difference is mainly due to different software implementations and features. Nowadays this trend not only holds for general computers, laptops and tablets, but also for embedded systems and cyber-physical systems.

Such systems are more and more used for safety critical applications in domains like avionics and automotive. Good examples are the various "X-by-wire" systems which increasingly replace traditional mechanical connections. For obvious reasons, one must ensure the dependability and reliability of such systems before they can be put in use.

Unfortunately the increasing complexity resulting from ever more functionality realized in software renders safety analysis more difficult. In contrast to mechanical systems, the effects of failures in software are not continuous. For physical devices, most often a small failure will have rather small effect; in contrast to that, even a small software error can have completely unpredictable effects. In addition, it is not sufficient to verify software in isolation of its environment, as it is done in classical software verification. In any case, the effect of possible hardware errors must be taken into account, as well as the physical environment where the system will be used. Finally, even if an accurate analysis and evaluation of an embedded system for a safety critical application is possible, it is not obvious if this is the best possible system variant with specific properties. In general, different and often even antagonistic objectives must be considered for the final system configuration.

A lot of work has been done to tackle the problem of qualitative safety analysis for complex embedded systems where component failures must be taken into account, *e.g.*, see [39, 7, 6, 34, 12, 41]. These approaches allow for analysis of combinations of component failures which may cause a potentially dangerous system malfunction, called a hazard. For certification of a safety critical system, it is important to conduct a quantitative safety analysis computing the hazard occurrence probability. This is often achieved using rather approximations based on qualitative analysis results and requires assumptions of stochastic independence. Newer approaches like [2, 5, 17] compute hazard probabilities directly, with a much higher degree of accuracy, but also with higher computational cost. There also exist some first approaches to optimize systems like [29, 37] which try to find a system variant which is optimized, in order to improve its reliability.

The approach presented in this article consists of the following: we propose to use the safety analysis and modeling language (SAML) [16] to model safety critical systems, allowing for qualitative and quantitative safety analysis on the same model, *i.e.*, without the need to construct different models for each of the different analyses. We extend SAML to support the specification of a set of possible systems and of different objective functions for optimization. Modern multi-objective optimization algorithms, including means to reduce computation time, are used to find the best possible system variants. All analyses are conducted fully automatic using different state-of-the art model checking tools. The approach is tool-independent, it uses model transformations to convert SAML models into the input specification language of analysis tools which allows for easy integration of new analysis tools. All this is integrated into an extensible framework based on the Eclipse platform. The work presented here is an extension of our first, rather ad-hoc approach to safety optimization, presented in [18]. We now have developed an explicit notion of variability modeling and provide an interface for arbitrary objective functions and for arbitrary estimation methods. We also present the results of our experiments using different approaches to exploit estimation techniques to make the optimization process more efficient.

## 2. Modeling

Our approach is based on creating a model of the system with its intended functional behavior, the behavior of its surrounding physical environment and the occurrence pattern and effect of failure modes.

We use a rather low level modeling language to express our models. The safety analysis and modeling language (SAML) [16] is derived from the PRISM modeling language [31] and describes Markov decision processes (MDP) [9]. This formal model allows the combination of software modeling, where failure modes are often non-deterministic and physical component modeling, where failure modes are often probabilistic. It is also possible to combine per-time and per-demand failure mode modeling with high accuracy [17]. Model transformations allow the analysis with different verification tools, dependent on the desired properties.

We chose SAML, as it is very close to the underlying formal model and therefore does not introduce much overhead into the state space. Nevertheless, it is possible to transform higher-level models to SAML for analysis. For an outline of that approach see [19] and some discussion of such possibilities in Section 6. The following description of SAML is adapted from [15].

### *2.1. MDP/SAML*

The most important aspect in the development of SAML was the possibility to model the control software, the physical environment and possible failure modes. These are all relevant for embedded systems in a safety critical domain. A SAML model is then analyzed using state-of-the-art verification engines, using proven correct model-transformation. In most cases, a SAML model consists of a model of the nominal behavior, an accurate model of the behavior of its physical environment and probabilistic failure mode modeling. Non-deterministic behavior facilitates specification of software failure modes and environment modeling, where probabilities are not known or cannot be given. Probabilistic behavior often reflects well physical environment modeling with known probabilities and failure modes of physical components.

Syntactically, SAML models describe sets of finite state automata with non-deterministic and probabilistic transitions. These are executed in a synchronous parallel fashion with discrete time-steps. SAML has been designed to be tool-independent and as simple as possible, while being expressible enough for convenient modeling of larger case-studies. It has been implemented using the ANTlr framework with an Eclipse-based specification front-end. Automatic model transformations for well-known model checkers (NuSMV, PRISM) also exist. For space restrictions, we do not present the complete formal syntax and semantics here. The complete definitions can be found here [16].

The grammar rules for the syntax of the most important part of SAML is shown in Figure 1 in Extended Backus Naur Form (EBNF) notation [1]. The

[1] Lexer rules are written uppercase, parser rules in lowercase. Bold font indicates keywords

actual implementation is done using the ANTlr parser generator [38]. The syntax is derived from the input language of the PRISM model checker [26, 31]. The main differences to the PRISM language are the absence of synchronization labels and the explicit modeling of non-deterministic choices with the `choice` keyword. These changes were done to facilitate the correct constructions for formal safety analysis. It forces the user to adhere to the modeling guideline which results in SAML models for which the presented safety analysis is sound. In contrast, PRISM uses implicit non-determinism modeling, *i.e.*, overlapping activation conditions. In the case of only a partial overlap, the probability distributions must be normalized which potentially changes the intention of the user. Therefore we chose to make non-determinism modeling explicit and treat overlapping activation conditions as modeling error.

```
saml-model :  (constant | formula)* module+ ;
constant :  constant TYPE IDENT (:= value)?  ;
formula :  formula IDENT := condition ;

condition :  ( condition )
             | !  condition
             | condition & condition
             | condition | condition
             | term ;

term :  IDENT (= |< |> |>= |<=) state_expr
        | IDENT | true | false;

module :  module IDENT declaration+ update+ endmodule;
declaration :  IDENT : [ INT ..  INT ] init INT ;

update :  condition -> non-det_assigns
          | prob_assigns ;

non-det_assigns :  non-det_assign
                   (+ non-det_assign)* ;

non-det_assign :  choice ( prob_assigns ) ;
prob_assigns :  prob_assign (+ prob_assign)* ;

prob_assign :  probability :  nextstate_assign
                              (& nextstate_assign)* ;

probability :  IDENT | DOUBLE | arith_expr;
nextstate_assign :  ( IDENT' = state_expr ) ;

state_expr :  IDENT | INT | ( state_expr )
              | state_expr (+ | - | / | *) state_expr ;

arith_expr :  INT | DOUBLE | IDENT | ( arith_expr )
              | arith_expr (+ | - | / | *) arith_expr
```

Figure 1: Basic SAML syntax

Figure 2 shows an example SAML model, consisting of two modules, `A` and `B`. Module `A` contains the state variable `V_A` with a value range from 0 to 2 and

or literal symbols without explicit lexer rules.

an initial value of 0. Module B contains 2 state variables, both with a value range from 0 to 1 and an initial value of 0. In module A, there are 5 updates rules which assign new values to the state variables for the next time-step. Module B has only 1 update rule, but with two non-deterministic choices. For both modules, every choice contains probabilistic transitions with parallel assignments for all state variables contained in the respective module.

```
constant double P_A := 0.1;
constant double P_B1 := 0.2;
constant double P_B2 := 0.3;
constant double P_B3 := 0.5;
formula CASE_3:=V_A=0& !(V_B1=0&V_B2=0|V_B1=1&V_B2=1);


module A
V_A:[0..2] init 0;
   V_A=0&V_B1=0&V_B2=0 ->
    choice (P_A:(V_A'=0)+(1-P_A):(V_A'=1));
   V_A=0&V_B1=1&V_B2=1 -> choice (1:(V_A'=2));
   CASE_3 -> choice (1:(V_A'=1));
   V_A=1 -> choice (1:(V_A'=1));
   V_A=2 -> choice (1:(V_A'=2));
endmodule


module B
V_B1:[0..1] init 0;
V_B2:[0..1] init 0;
true -> choice (P_B1:(V_B1'=0)&(V_B2'=0) +
     P_B2:(V_B1'=1)&(V_B2'=0) +
     P_B3:(V_B1'=1)&(V_B2'=1)) +
    choice (1:(V_B1'=1)&(V_VB2'=1));
endmodule
```

Figure 2: Example SAML model

The informal semantics of the above example is as follows: both modules are executed synchronously parallel, *i.e.*, at each time-step all variables from both modules get a new value assigned, which is specified by the updates rules. Every update rule has a Boolean activation condition. When this condition holds, the update rule is active and specifies the next values for all state variables of the module. At every time-step *exactly one* activation condition holds, *i.e.*, there is no termination state and there are never two active updates (this can be checked automatically). Every update specifies a non-deterministic choice of discrete probability distributions, as indicated by the `choice` keyword. Each probability distribution specifies a parallel assignment of new values to the state variables and its probability.

Initially the state variable `V_A` has the value 0, `V_B1` the value 0 and `V_B2` the value 0, therefore the activation condition of the first update of `A` holds. It consists of a single non-deterministic choice, for which the probabilistic distribution assigns the value 0 to `V_A` with probability $p_A$ and the value 1 with probability $1 - p_A$. For the module `B` there is only one update with activation condition `true` which is therefore active at each time-step. The non-deterministic choice is between two probability distributions. Each probability distribution is a par-

allel assignment of new values to the two state variables of `B`. The first assigns the values $(0,0)$ to `(V_B1,V_B2)` with probability $p_{B1}$, $(1,0)$ with probability $p_{B2}$ and $(1,1)$ with probability $p_{B3}$. The second assigns $(1,1)$ with probability 1, *i.e.*, deterministically.

The formal semantics of SAML is a labeled Markov decision process (MDP) which results from computing the product automaton of the parallel SAML. This parallel composition, derived from [31], constructs a single SAML module with all state variables and combined update rules from the original model, for details see [16]. More details on the semantics of the underlying MDP can be found for example in [9].

### *2.2. Failure Mode Modeling*

In general, SAML offers the means to specify a Markov decision process in the form of synchronous parallel finite state machine. For our quantitative safety analysis approach we use an explicit model of the occurrence of failure modes. Our modeling allows a combined use of per-demand and per-time failure modes. The occurrence of a per-demand failure mode is triggered by an event and the failure can only occur at that moment, *e.g.*, a braking of a handle. Such failures are specified via a failure occurrence probability.

An example for a persistent, *i.e.*, non-repairable, per-demand failure mode is the following:

```
pd_persistent_failure_module pdpfm1
      demand demand := action;
      failure_probability := p_f;
endpd_persistent_failure_module
```

Here *demand* is a the name of the Boolean formula *action*. It models the intent to activate the safety critical component. This fails with probability $p\_f$ and succeeds with probability $1 - p\_f$. If the failure mode occurs, the state of the failure modules changes to signal this occurrence.

In contrast to this, a per-time failure mode does not have a specific trigger, but may occur at any time, *i.e.*, a sensor reading failure. Such a failure is always specified in relation to a time interval. For this we use the notion of a global temporal resolution $\Delta t$ in SAML models, defining the amount of time that passes at each time-step. An example for a per-time failure mode is the following:

```
pt_persistent_failure_module ptpfm1
      failure_rate test_pt_pers := 0.06 per min;
endpt_persistent_failure_module
```

This models the occurrence of a transient failure mode with a failure rate of $0.06\frac{1}{min}$.The usage of an explicit notion of the temporal resolution allows for

a good approximation of continuous time behavior, in particular for the per-time failure modes [17]. For more detail on failure mode occurrence and effects modeling see [32]. The specific goals of using this explicit notion of the failure mode modeling are the following:

- **Adequacy of the model:** It reduces many possible causes for specification errors. In particular, computations of adapted failure rates for different temporal resolutions is not necessary. The temporal resolution has a great impact on the accuracy, but also on the required analysis time of the model [33]. Varying the temporal resolution can be useful to have a quick first approximation, *i.e.*, for a certain system variant (see Section 3 for more discussion on that). It facilitates the understanding of the modeling choices of specific failure modes, helping to judge the adequacy of the model.

- **Flexibility:** During verification, models often have to be "trimmed" manually to become analyzable. One common way to do this is by coarser abstractions. Changing the temporal resolution of a SAML model provides an easy way to achieve this, as there is only a single point requiring change[2]. This makes the models flexible, as different resolutions of the model are comparable without much effort. However, note that currently probabilistic models of the environment situation (*e.g.*, how often are certain requests given to the system) might require manual changes (further support in this area is planned in the future).

- **Functional Correctness:** With this modeling, there is an explicit notion of failure mode behavior in the specification. This allows for (i) full automatisation of some model-based safety analysis methods (by automatic generation of the failure mode dependent proof obligations), (ii) automatic verification of a conservative failure mode integration by analyzing the model on a *syntactic* layer and (iii) generation of a system model without failure behavior (*i.e.*, removing transitions or states addressing the direct effects of failure modes) from the model. The last option will typically make the model smaller and thus possibly analyzable with other tools (although verification is then limited to functional correctness).

### *2.3. Variability Modeling*

In this paper we extend the existing analysis of a single SAML model to the search for best compromises wrt. different, antagonistic goals. This requires the extension of the model description from a single one to a parametric system model which represents a whole set of different variants.

We propose a well formed extension to SAML that allows for the specification of families of models. Such a family consists of a set of different concrete system

[2]Other than temporal abstractions will in general also require changes in functional parts of the model. These must of course still be done manually.

implementations. Alternative systems are expressed by means of parametric constants and alternative modules, i.e., a family is implicitly expressed as an under-specified system where certain details are variable. A specific element of such a family is then instantiated by the specification of concrete values for all parametric constants and a concrete selection of alternative modules. The system family is not fully generated, the proposed optimization (see Section 3) tries to find the optimal set of parameters by informed sampling.

On the syntactical side we introduced the new keyword *param* as an extension to the SAML language as described in [16]. It can be used in the context of constants and modules where it has the following meaning:

**param constant** Constants are named labels for fixed numbers in a model. Certain models may differ in the values of one or more constants. A *param constant* enables the specification of constants as a closed interval of numbers instead of a single specific value. Param constants may be, similar to constants, of integer and floating point types.

**param module** Different variants of a system may feature different specifications of certain components. We reflect these variants by specifying interchangeable modules.

We named the new variant of SAML with the *param* keyword optimizable safety analysis modeling language (OSAML). The additions to the SAML grammar are listed in Figure 3.

```
rule :  (metaconstant | formula)* metamodule+ EOF

metaconstant:  constant | paramconst
metamodule:  module | parammodule

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

Figure 3: The OSAML grammar contains only a few additional and/or different rules than the SAML grammar.

The semantics of an OSAML model is informally defined as the family of SAML models which can be constructed from the parameter space. Every member of this family is a complete and valid SAML model. Thus, the variability is explicitly expressed by the model creator (i.e. the engineer). Such a model family may consist of good and bad candidates wrt. different objectives. As the optimization focuses on the identification of good candidates, only members of the family can be analyzed and it is not possible to analyze a whole family at once. We call the selection of a specific member out of a OSAML model the instantiation of the model.

The instantiation of a OSAML model consists of two steps. Firstly, a single value must be assigned to each parametric constant which is feasible wrt. the

param specification. This constant remains as a simple constant in the generated SAML model. Secondly, a single feasible module must be chosen for each parametric module. This module remains as a normal module in the generated SAML model. The instance is then a SAML model without variability and it can be analyzed with the existing techniques, *i.e.*, model transformations and model checking.

### *2.4. Objective functions*

We perform the optimization of probabilistic models along multiple, possible antagonistic, goals. These goals (also called objective functions) assign a quantitative value to every model variant. This allows for the comparison of two or more variants with each other. Our framework supports two different types of objectives out of the box. First of all, and most important, objectives can be given as a PCTL probabilistic temporal logic [20] formula. Secondly, a generic Java interface enables expression of arbitrary objectives.

In this section we first provide an informal introduction to PCTL and then introduce the Java implementation of generic objectives, *i.e.*, any function which maps the model parameters to a scalar value and implements the specified interface.

#### *2.4.1. Quantitative Model Checking Based Objectives*

The PCTL formalism is a probabilistic extension for the computation tree logic (CTL). CTL [8] is a branching time temporal logic, it provides the modal operators **A** (on all future paths) and **E** (there exists a path) which state whether the property holds on one or all possible future paths. The modal operators are combined with one of the four temporal operators: **X** (in the next step), **F** (eventually), **G** (globally) and **U** (until). The combination of modal and temporal properties allows the expression of qualitative model properties such as: If there exists a path on which a given propositional expression $\phi$ is always true ($\mathbf{EG}\phi$) or if a given $\phi$ will eventually hold on all future execution paths ($\mathbf{AF}\phi$).

For safety analysis, the reachability of hazardous states is of particular interest; such an analysis is also performed for the case study in Section 5.1. We express the possibility to reach a state for which a propositional formula $\phi$ holds which describes the hazard as $\mathbf{E}[true\ \mathbf{U}\ \phi]$. Informally this means that there exists a path for which *true* holds until $\phi$ becomes true, i.e., it is possible to reach one of the hazardous system states.

CTL formulas allow only a qualitative analysis of a model. In addition to that, PCTL formulas exploit probabilistic information of a model. Besides the question if a certain formula is satisfiable, PCTL allows the expression of restrictions on probabilities. In simple terms, PCTL replaces the modal quantifiers with a probabilistic quantifier $P_{\sim p}$ where $\sim \in \{<, >, \leq, \geq\}$. It computes whether for the probability $p'$ (i.e., that the specified property is true on a randomly selected path) the condition $p' \sim p$ holds. A useful extension of PCTL allows to compute directly the probabilities of a given temporal property.

For models expressed as Markov decision processes (MDP) [9], it is impossible to compute exact probabilities of PCTL formulas. Instead the probabilistic quantifier $P_{min}$ and $P_{max}$ are used, stating the minimal/maximal property that the subsequent temporal formula [3] holds, which is often interpreted as the best and worst case outcome. This is due to the nature of MDPs, for which transitions can be both non-deterministic and probabilistic.

In this paper we concentrate on safety related objectives, i.e., the probabilistic deductive cause consequence analysis (pDCCA) [16]. This leads to formulas expressing the maximal probability to reach a certain set of states (defined as Hazard by a propositional formula $H$), i.e., the highest probability to reach a hazardous state, assuming worst case non-deterministic choices.

$$P_{max=?}\ [true\ \mathbf{U}\ H] \tag{1}$$

However, the proposed optimization framework is not limited to these specific formulas. Any PCTL formula can be used directly, other quantitative logic formulas also with proper tool support.

*2.4.2. Java Based Objectives*

Besides the model checking based objectives we provide a Java interface for arbitrary objective functions. This is a simple and powerful way to specify cost and performance based objectives, i.e., objectives which are given as mathematical formulas over the optimization parameters. Expressing an arbitrary objective is as simple as implementing the *IObjective* interface. This Java interface defines the functions *evaluate* and *getResult*. The evaluate function takes the parameters of a certain model and the model instance itself as parameter. The results of the objective evaluation are returned by the getResult method. The entire software architecture of the framework is described in Section 4.2.

In addition to mathematical formulas, the Java interface enables the integration of more sophisticated analysis algorithms. As a matter of fact the quantitative model checking based objectives described in the previous section is a function which instantiates the *IObjective* interface, runs the PRISM model checker, parses its output and returns this as result.

## 3. Optimization

In general, safety critical systems are developed using the principle to have the risk *as low as reasonably possible* (ALARP). So in order to develop a good system, increasing the safety of a system will always be an important goal.

On the other hand, it must be kept in mind that safety is never the only goal, any system which should be realized must also be functional and economical. This is captured in the "reasonably" part of ALARP, for which there is no universally valid definition.

In order to help in the development of safety critical systems, we propose a safety optimization approach which allows for optimizing a system for several

objective functions at the same time using multi-objective optimization techniques [18]. As a special case, it is possible to limit the optimization to safety, *i.e.*, to minimize only the hazard probability of a system.

In this chapter we will first present a generic technique for multi-objective optimization based on genetic algorithms. After this we will present means to use estimation to speed up the evaluation of very time-intensive goal functions and conclude the chapter with a discussion of how optimization and estimation can sensibly be combined for efficient safety optimization.

### *3.1. Genetic Algorithms and NSGA-II*

Genetic algorithms [4] are an optimization method which is generally applicable, it tries to emulate the biological evolutionary process.. In particular, genetic algorithms treat the objective functions as black box functions. The only requirement is to evaluate the objectives for a given set of parameters. In contrast to analytic approaches the evolutionary algorithm do not need additional insight information like higher order derivative of the objective functions. An advantage of this is that advanced safety analysis techniques like pDCCA can be used as optimization objective, the disadvantage is that often many function evaluations are required to get good convergence.

The general scheme works as follows: At first, a set of solution candidates called the population is created. In the population every solution candidate represents a possible element of the search space. Every candidate is evaluated using the goal functions and its fitness is compared to the other solution candidates. Good solution candidates are then taken from the population and are combined to create a potentially better new solution. To explore the search space, some random changes in the solution candidates are also executed which are called mutation.

There are many different schemata to rank the solution candidates for their fitness, how to combine them to create new candidates, how to mutate for exploration and how a new population is created.

One of the most widely used multi-objective optimization algorithms is the Non-Dominated Sorting Genetic Algorithm in its version 2 (NSGA-II) [11]. We chose to use it, as it is well suited for both integer and real valued parameters using its SBX combination operator [10] and provides means to guarantee good exploration of the search space. Nevertheless, any other general multi-objective optimization algorithm is applicable to our safety optimization approach.

NSGA-II generally works as follows: First an initial population of solution candidates is generated. Using Latin-hypercube sampling, we try to sample a representative part of the whole search space. After that, the function values for each solution candidate are computed. The fitness function is then computed, incorporating not only the functional values, but also a measure for solution candidate "crowding", assuring diversity in the population. The exact details can be found in [11]. The selection of the candidates for combination with the SBX operator is done using Roulette selection. In this way the better a candidate is ranked, the higher its probability for selection. On the other hand

even low ranked candidates have a certain probability of being chosen. After the recombination, some of the candidates are randomly mutated and a new iteration begins.

### *3.2. Estimation Algorithms and ANN*

The run-time performance of evolutionary algorithms highly depends on the costs of the objective function evaluation. They need many function evaluations to find Pareto optimal parameters, the more time a single function evaluation requires, the longer the overall run-time of the algorithm will be. In the literature on genetic algorithms, there is often the assumption that the evaluation cost of objective functions is negligible. This is in stark contrast to the probabilistic model checking objectives we use in this paper, where a single function evaluation, *i.e.*, run of the model-checker, can easily require several minutes or even hours for large models.

However, multi-objective evolutionary algorithms are a statistically guided search technique. This suggests that precise function evaluations are not necessary all the time, but good estimations could be used which guide the search. In our approach we treat the objective functions (especially the model checking based ones) as black-box functions, thus it is not necessary to derive the estimation from the objective functions itself.

In summary this leads to two major requirements for an appropriate estimation technique: (1) It must be adaptable, i.e., a set of real function evaluations must augment the estimation algorithm. (2) The estimation algorithm must be able to estimate arbitrary real valued functions. We choose artificial neural networks (ANN) as estimation algorithm, as they generally meet both requirements. They can be trained by examples and also have the ability to estimate arbitrary continuous and real valued functions [21].

ANN are inspired by nature and try to emulate the functionality of biological neural networks. Simply put an ANN is a network of similar and trivial computational units. Each computational unit emulates the behavior of one biological neuron. It is modeled in such a way, that its output signal is the result of a so called activation function calculated for the weighted sum of all inputs. The activation function usually is a sigmoid or linear function. The only parameters of every neuron (values that define the knowledge of the network after training) are the weights of the inputs of all neurons. In general, the ANN concept allows arbitrary connections between all the neurons. In this application we limit to multilayer feed forward network structure. This is the most widely used variant of ANN.

For training we use the improved resilient propagation algorithm, which is a variant of backpropagation learning algorithm [23]. The training data sets (i.e., pairs of input and corresponding expected output values) are fed to the network and the estimation error of the network is calculated. The weights of the output layer neuron are then altered in such a way that the error is minimized and the error on the output is propagated one layer backwards to the first hidden layer. Again the weights are altered and the error is propagated backwards. This is repeated until the weights of all neurons were adapted. Then this procedure

is repeated with the next training data set. The training of the network, once with every training data pair, is called an epoch and the training is finished after a certain maximum number of epochs or if the average error falls below a threshold.

### *3.3. Combining NSGA-II and ANN*

In the last two subsections we explained the used optimization and estimation algorithms. The remaining question is how the two are interconnected. In Figure 4 we show our estimation strategy. In the first step we use the selection and crossover operators to create an offspring population. Instead of the function evaluation of all newly created individuals we then use the estimation function and the non-dominated sorting (as defined for the NSGA-II algorithm) to drop the weaker half of the offspring population. The remaining candidates are then evaluated with the true objective functions. From here on the normal NSGA-II algorithm continues: The offspring and the current population are combined and non-dominated sorting is used to select the candidates for the next generation.

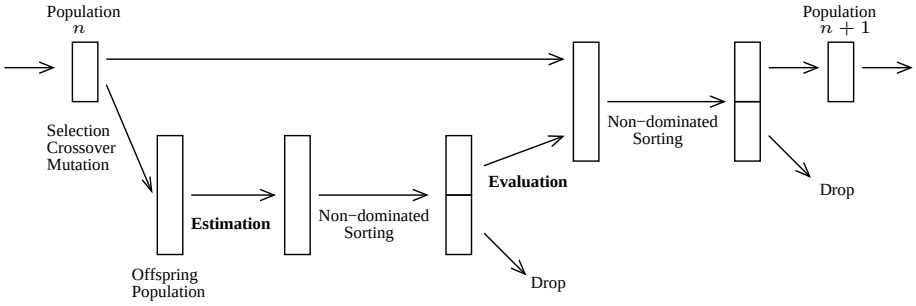



Figure 4: Estimation Strategy

In our approach we store all individuals that were ever evaluated throughout the optimization procedure in a central storage. This pool is used in every generation as reference data to train the ANN.

## 4. Tools

The optimization approach is based on set of external tools and libraries. In this section we first introduce the most important ones and then describe the software architecture of the implemented optimization framework.

### *4.1. Used Tools and Libraries*

We used the S$^3$E [28] to create the SAML models. It is an Eclipse based tool for formal systems specification and verification we developed. It provides an editor for SAML and has the ability for step-wise simulation of models. However, the use of S$^3$E is mainly motivated by convenience of specification

and analysis. The resulting SAML/OSAMLmodel is stored in a plain text file, which can be created with any text editor as well.

An important tool for the optimization framework is ANTLR [38], a framework for the creation of language recognizer, compiler, interpreter and translators. We used ANTLR to create a parser that turns a OSAML model into an abstract syntax tree representation (AST) which is used to instantiate a single SAML model. The AST representation of the SAML model is then transformed into the input language of the desired model checker using model to text transformations.

The artificial neural network estimation is built upon the Fast Artificial Neural Network Library (FANN) [30]. It provides a C implementation of multi-layer feed forward networks and supports both sparsely and fully connected networks. For the training of the network, FANN uses improved resilient propagation (iRPROP) [23]. To integrate the C library into the Java runtime environment, we used the FannJ[3] interface.

The qualitative safety analysis of a SAML model is performed with the probabilistic model checker PRISM [27]. It supports the probabilistic analysis of discrete time Markov chains, Markov decision processes, probabilistic timed automata and continuous time Markov chains. We integrated the PRISM model checker into the optimization process by creating the specification and probabilistic temporal logic property files, invoking PRISM and parsing the textual output of the model checker.

### *4.2. Software Architecture*

The simplified architecture of the optimization framework is depicted in Figure 5 as UML class diagram. It starts with the *IOptimizer* interface which is implemented by the *NSGA2* class and the *BruteForce* class. The first one contains the NSGA-II genetic algorithm and the second one contains a brute force approach. The latter one enumerates the whole search space and filters Pareto optimal candidates. Due to its nature it can only be applied to problems with rather small and enumerable search spaces. However, due to the *IOptimizer* interface, the framework is easy to extend and the appropriate algorithm can be used for every optimization problem.

The implemented optimizers use the *SamlSystemCluster* class which covers OSAML models (i.e., a family of SAMLmodels). The *generateInstance* method is used to select a specific SAML model out of the model family. A SAML model is represented by the *SamlSystem* class.

The OSAML model is represented by the *SamlSystemCluster* class. The SAML models which are members of the OSAML model family are represented by the *SamlSystem* class. The system cluster is responsible to create *SamlSystem* instances according to specific parameters choices.

Every objective must implement the *IObjective* interface to provide an *evaluate* method. The *PrismObjective* is a generic objective which is configurable

[3] `http://code.google.com/p/fannj/`

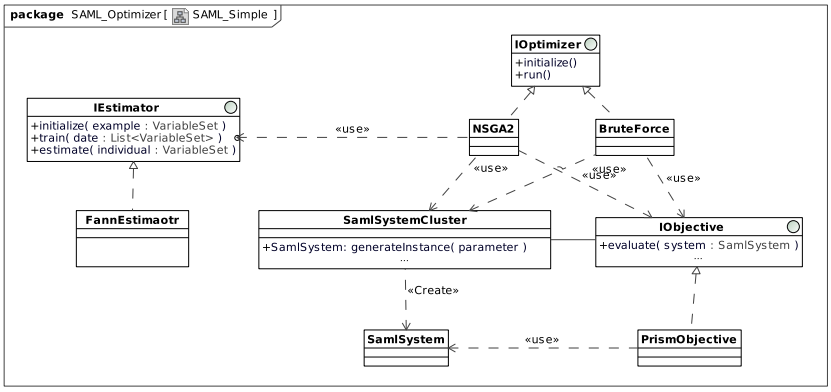



Figure 5: Simplified software architecture of the optimization framework.

with a PCTL property. It covers the evaluation of a SAML model with the PRISM model checker according to the specified PCTL property. All other objectives are integrated according to the optimization problem by implementing a Java class that implements the *IObjective* interface.

On the function estimation side we defined a separate interface. This allows the easy exchange of different estimation algorithms. However, at the moment we only provide a realization based on the FANN library.

## 5. Experiments and Results

### *5.1. Case Study*

The following case study was proposed by the German railway organization, Deutsche Bahn. It was used as a reference case study in the priority research program 1064 "Integrating software specifications techniques for engineering applications" of the German Research foundation (DFG). Its purpose was to analyze an alternative scenario for railroad crossings where the control is completely decentralized and based on direct radio communication between the approaching train and the barrier. It was aimed at medium speed routes with a maximum speed of $160\frac{km}{h}$.

The following description of the case study is taken directly from [40]:

> "The main difference between this technology and the traditional control of level crossings is, that signals and sensors on the route are replaced by radio communication and software computations in the train and in the level crossing. This offers cheaper and more flexible solutions, but also shifts safety-critical functionality from hardware to software.
>
> Instead of detecting an approaching train by a sensor, the train computes the position where it has to send a signal to secure the level crossing. Therefore the train has to know the position of the

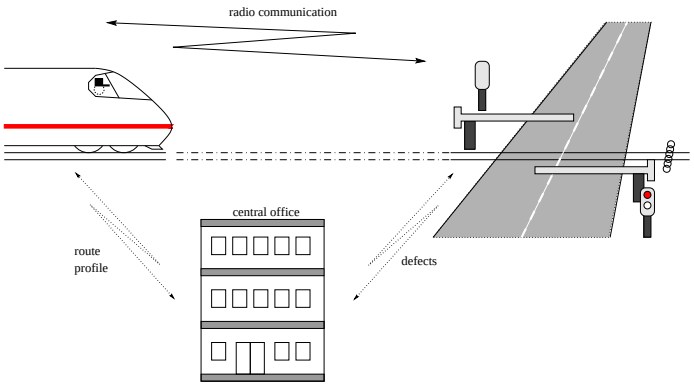



Figure 6: Radio-based Railroad Crossing [40]

level crossing, the time needed to secure the level crossing, and its current speed and position, which is measured by an odometer.

When the level crossing receives a *secure* command, it switches on the traffic lights, first the *yellow* light, then the *red* light, and finally closes the barriers. When they are closed, the level crossing is *safe* for a certain period of time. The stop signal, indicating an insecure crossing, is also substituted by computation and communication. The train requests the status of the level crossing. Depending on the answer the train will brake or pass the crossing. The level crossing periodically performs self-diagnosis and automatically informs the central office about defects and problems. The central office is responsible for repair and provides route descriptions for trains. These descriptions indicate the positions of level crossings and maximum speed on the route."

To calculate the activation point, the train uses data about its position, its maximum deceleration and the position of the crossing. For safety reasons a safety margin is added to the activation distance. This allows for the compensation of some deviations in the odometer.

On a first sight, the primary objective is of course to minimize the risk of having a train passing the opened barrier. Nevertheless there are also other important objectives: the increased time delay induced by a too large safety margin can become relevant, as it prevents the usage of the rails for succeeding trains and will impact the schedules. Therefore it will also be a goal to minimize the additional time delay which is influenced by both the safety margin and the allowed speed of the train. The third important objective is of course the cost of the system. We allowed for variation in the exactness of the velocity measurement, making it directly proportional to the cost of such a system.

We modeled six different relevant failure modes for this case-study: *error_passed* models a sensor misdetection that the train has already passed the

crossing, *error_odo* models a deviation of the measured velocity in contrast to the real velocity, *error_comm* models a failure of communication between the train and the crossing, *error_close* models a wrong closed signal from the crossing, *error_actuator* models a stuck barrier while closing, and *error_brake* finally models a failure of the braking system of the train.

We modeled *error_comm* as a per-demand failure mode with an occurrence probability of $3.5 \cdot 10^{-5}$. The other failure modes (except *error_odo*) were modeled as per-time failure modes with a failure rate of $7 \cdot 10^{-9} \frac{1}{s}$.

For the optimization, we created an OSAML model of the case study which describes a family of SAML models of the railroad crossing with 3 variable parameters: The accuracy of the odometer measured as the failure rate $\lambda_{odo}$ of the deviation from the real velocity, the safety margin $z$ and the allowed maximum speed $v_{allowed}$ of the train when approaching the crossing. The odometer has a direct influence on the computation of the activation point. The safety margin adds a buffer to the distance of the crossing to the activation point and the velocity directly influences the time delay. As domain of the parameters we used the interval $[0, 1]$ for failure rates of the odometer, the interval between 0 and $200m$ for the safety margin and for the maximal velocity the values between 1 and $120\frac{km}{h}$. The objective functions considered in this case study were then formalized as:

$$
\begin{aligned}
f_1(\lambda_{odo}, z, v_{allowed}) &:= P[true \mathbf{U} H] \\
f_2(\lambda_{odo}, z, v_{allowed}) &:= cost(\lambda_{odo}) \\
f_3(\lambda_{odo}, z, v_{allowed}) &:= \frac{z + x_{brake}(v_{allowed})}{v_{allowed}} - \frac{z + x_{brake}(v_{allowed})}{v_{max}}
\end{aligned}
$$

The function $f_1$ uses pDCCA to compute the occurrence probability of the hazard, *i.e.*, that the train passes the crossing which is not closed. The second function $f_2$ models the cost of a more accurate odometer as being exponential in the inverse of the failure rate. The third objective function $f_3$ computes the time delay when comparing the reduced velocity of the train with its normal average velocity on the track. This means that the evaluation of $f_2$ and $f_3$ for a given set of parameters is rather quick, in contrast to the evaluation of $f_1$ which requires a formal quantitative safety analysis, i.e., a run of the probabilistic model checker.

### *5.2. Evaluation Methodology*

Many evolutionary algorithms advance in terms of generations. Therefore, concerning the estimation algorithm, it seems natural to use the information gathered from the function evaluation of the first $k$ generations to estimate generation $k + 1$. We chose to use artificial neural networks (ANN) for this estimation, as they are a general and widely applicable learning and estimation method. Nevertheless, their usage integrated into our optimization approach raises several questions:

- How good does an ANN perform in estimating a quantitative safety objective?
- How many function evaluations are required to provide sufficient reference data for the network training?
- What are the best parameters for the network (i.e., number of layers and number of neurons per layer)?

To evaluate the usage of ANN in combination with the evolutionary algorithms we followed a two-step approach. In the first step we analyzed the performance of various ANN configurations to estimate the safety objective function. After that we combined the estimation algorithm with the optimization and compared the results of the simple optimization with the results of the estimator enhanced optimization.

*5.2.1. Evaluating the ANN Estimators*

In the first step we trained and evaluated the ANN on pre-calculated data from an optimization process that was performed without estimation algorithm. The main intention of this experiment was to evaluate if ANN are sufficient to estimate the PRISM-based objective functions when being trained with the data of subsequent generations. The basic idea is to train the ANN on the data from $k$ generations of function evaluations. After that the ANN is used to estimate the generation $k+1$. The error made by the estimator is determined by comparing the estimated value with the value of the actual function evaluation.

We call $\mathbf{x}_j^{(k)} \in S$ the parameter vector of the $j$-th candidate in generation $k$. $S$ is the space of feasible parameter values. The objective function is defined as $f : S \to \mathbb{R}$. Consequently $f(x_j^{(k)})$ denotes the value of the objective function in generation $k$ for candidate $j$. The function $g^{(k,l)} : S \to \mathbb{R}$ is the estimation function for the objective function that was trained with the reference data of generation $0 \dots k$. As the training of the ANN starts from a random initial configuration, we repeat the training and estimation of one generation multiple times. The superscript $l$ counts the repetitions that the ANN was trained for that specific generation.

As a measure of the error made by the estimator we use the mean of the absolute estimation error of all candidates in one generation.

**Definition 1. Absolute estimation error** *The absolute estimation error of candidate $j$ in generation $k+1$ made by an ANN that was trained on generations $0 \dots k$ is defined as:*

$$e_j^{(k+1,l)} = \left| g^{(k,l)} \left( \mathbf{x}_j^{(k+1)} \right) - f \left( \mathbf{x}_j^{(k+1)} \right) \right| \tag{2}$$

**Definition 2. Mean estimation error in a generation** *The mean estimation error in generation $k+1$ of size $J$ made by an ANN that was trained $l$*

*times on generations* $0 \dots k$ *is defined as:*

$$\hat{e}^{(k+1,l)} = \frac{1}{J} \sum_{j=1}^{J} e_j^{(k+1,l)} \tag{3}$$

Due to randomized initial values in a newly created ANN, the success of the training and the subsequent estimation error cannot be measured with only one ANN with one training cycle. To overcome this obstacle, we trained $L$ equal ANNs and calculated the average of all $L$ mean estimation errors in one generation:

**Definition 3. Average estimation error in a generation** *The average estimation error in generation* $k+1$ *made by* $L$ *independent ANN of similar type which where all trained on generations* $0 \dots k$ *is defined as:*

$$\hat{e}^{(k+1)} = \frac{1}{L} \sum_{l=1}^{L} \hat{e}^{(k+1,l)} \tag{4}$$

The safety objective used in the case study as well as one of the parameters are probabilistic values. They are defined on the interval $[0, 1]$. Within the context of estimation, their magnitude is of more importance than their exact value. This is especially important for the training of the ANN. The iterative training process is stopped when a maximum number of iterations is reached or if the estimation error of the ANN on a certain training set falls below a given threshold $\epsilon$. Applied to probabilities, $\epsilon$ gives a lower bound to the smallest probability that can be estimated with feasible accuracy. At the same time, a small $\epsilon$ requires greater probabilities to be estimated with an unnecessary high accuracy. Thus, we applied logarithmic scaling to all probability values. On the parameter side the parameter $p_{failsOdo}$ was replaced with

$$p'_{failsOdo} = log_{10}(p_{failsOdo}). \tag{5}$$

On the objective side, the safety objective function was replaced with

$$f'(\mathbf{x}) = log_{10}\left(f(\mathbf{x})\right). \tag{6}$$

Consequently the same applies for the estimation function:

$$g'^{(k+1,l)}(\mathbf{x}) = log_{10}\left(g^{(k+1,l)}(\mathbf{x})\right). \tag{7}$$

This allows a good comparison of the magnitude of the probabilistic values. The logarithmic scaling in context of the difference based error measure leads to an error measure that is actually the logarithm of the relative error.

We tested the estimation algorithm on the results from three independent reference optimization runs, which were performed with the NSGA-II algorithm. Details on the reference optimization are provided in Section 5.2.2. For the experiments we used networks with 1 and 2 hidden layers and 5, 10 and 20 neurons

per hidden layer. To eliminate the influence of randomized initial weights in the network, we trained and estimated every generation 500 times and calculated the average error. All networks were fully connected feed-forward networks with a bias neuron in every layer. The hidden layer neuron had a sigmoid activation function and the neuron in the output layer had a linear activation function.

We present the results from the experiments with the ANN in section 5.3.1.

*5.2.2. Comparing the Optimization with and without Estimation*

For the comparison of the proposed estimation enhanced optimization with the plain NSGA-II based optimization we performed experiments with both algorithms in different configurations. We also refer to the NSGA-II based optimizations as *simple* and to the combination of NSGA-II with the estimator as *enhanced* optimization. All experiments are listed in Table 2. The first column states a name for every configuration. The second column lists the size of the population used for the optimization. In the *Generation* column the number of generations is listed. The last two columns list the number of hidden layers and the number of neurons in every hidden layer in the artificial neural network. We choose the number of generations and the population size in such a way that for every optimization a total of 480 objective function evaluations is performed.

| Name | Pop. Size | Generation | ANN-Layer | ANN-Neurons |
|---|---|---|---|---|
| opt-p16-g30 | 16 | 30 | - | - |
| opt-p16-g30-l1-n05 | 16 | 30 | 1 | 5 |
| opt-p16-g30-l1-n10 | 16 | 30 | 1 | 10 |
| opt-p16-g30-l2-n05 | 16 | 30 | 2 | 5 |
| opt-p16-g30-l2-n10 | 16 | 30 | 2 | 10 |
| opt-p24-g20 | 24 | 20 | - | - |
| opt-p24-g20-l1-n05 | 24 | 20 | 1 | 5 |
| opt-p24-g20-l1-n10 | 24 | 20 | 1 | 10 |
| opt-p24-g20-l2-n05 | 24 | 20 | 2 | 5 |
| opt-p24-g20-l2-n10 | 24 | 20 | 2 | 10 |
| opt-p32-g15 | 32 | 15 | - | - |
| opt-p32-g15-l1-n05 | 32 | 15 | 1 | 5 |
| opt-p32-g15-l1-n10 | 32 | 15 | 1 | 10 |
| opt-p32-g15-l2-n05 | 32 | 15 | 2 | 5 |
| opt-p32-g15-l2-n10 | 32 | 15 | 2 | 10 |

Table 1: List of configuration for all performed experiments

The great impact of random numbers in the evolutionary optimization algorithms requires a great number of repetitions of the experiments to gain sound results. As a matter of fact the enormous computational costs of the objective functions make it impossible to repeat the experiment more often within a feasible amount of time (i.e., several weeks of uninterrupted computation time on a modern multi-core computer for a single optimization). To cope with the im-

mense computation time of the PRISM based objectives we sampled the whole search space. We then used the pre-calculated data to perform the optimization experiments.

We sampled the search space at the following points:

$$p_{odo_{fails}} = \left\{ \begin{array}{llll} 1 \cdot 10^{-6}, & 2 \cdot 10^{-6}, & 3 \cdot 10^{-6}, & 5 \cdot 10^{-6}, \\ 1 \cdot 10^{-5}, & 2 \cdot 10^{-5}, & 3 \cdot 10^{-5}, & 5 \cdot 10^{-5}, \\ 1 \cdot 10^{-4}, & 2 \cdot 10^{-4}, & 3 \cdot 10^{-4}, & 5 \cdot 10^{-4}, \\ 1 \cdot 10^{-3}, & 2 \cdot 10^{-3}, & 3 \cdot 10^{-3}, & 5 \cdot 10^{-3}, \\ 1 \cdot 10^{-2}, & 2 \cdot 10^{-2}, & 3 \cdot 10^{-2}, & 5 \cdot 10^{-2}, \\ 1 \cdot 10^{-1}, & 2 \cdot 10^{-1}, & 3 \cdot 10^{-1}, & 5 \cdot 10^{-1}, \quad 9 \cdot 10^{-1} \end{array} \right\} \tag{8}$$

$$speed_{max} = \{1,\ 3,\ 5,\ 7,\ 9,\ 13,\ 15,\ 16\} \tag{9}$$

$$z = \{0,\ 2,\ 4,\ \ldots,\ 100\} \tag{10}$$

In contrast to the proposed optimization approach we evaluate the objectives based on the sample data. The evaluation is implemented as follows: If there exists a pre-calculated point for the given parameters, the result is used. In the more likely case that there exists no result for a certain parameter set, the nearest sample point is searched and the objective value for that sample point is used. The nearest distance is determined according to the Euclidean distance in the parameter space.

To gain reference data we extracted the Pareto front from all sample points made. This lead to a front consisting of 83 points. We show a graphical representation of the entire front in Section 5.3.2.

For all optimization experiments the crossover probability was 0.9. The mutation probability was 0.66 in the first 5 generations and from there on declining down to 0.1 in the remaining 15 generations. The initial values are chosen according to the publication of NSGA-II [11]. We chose to decrease the mutation probability to gain a good spread over the search space in the initial phase and to refine "interesting" regions in the later phases of the optimization. The training of the ANN was performed for a maximum of 10000 epochs or until the error[4] fell below 0.0001. The configuration of the ANN was based on an educated guess. We made a compromise between computation time and accuracy of the trained network. From the ANN point of view, all probabilistic values had a logarithmic scaling (see Section 5.2).

To evaluate the performance of the various different optimizer and estimation configurations we use two different metrics. In addition to the plain NSGA-II algorithm we stored every candidate that was ever evaluated in a central pool. After every generation, we extracted the front of Pareto optimal points from

[4]The error measure during training is the one given by the FANN library. See section 4 and [30]

the pool by non-dominated sorting. Based on the collected data we define the following measurements:

**Hypervolume indicator** Measures the size of the volume defined by a front (given as set of points) and a reference point [42]. For our experiments we used the Python implementation by Simon Wessing[5].

**Pareto-Count** Measures after every new offspring generation the number of all found Pareto optimal points (with respect to the reference front) found this far.

To gain statistical data we repeated every experiment (see Table 1) exactly 500 times. We then calculated the average value for all measurements on each experiment.

### *5.3. Results*

In this section we present the results of the evaluation of the estimation algorithm and the comparison of the simple and the enhanced optimization approaches.

#### *5.3.1. Evaluation of the Estimation Algorithm*

For the experiments we first trained the network on $n$ generations and then estimated in generation $n+1$. For every estimated generation we calculated the average error over all 500 repetitions. Due to space limitations we only show the results for one experiment in Figure 7. The upper curves denote the average estimation error and the lower curves denote the standard deviations. The other two experiments led to values in the same region as shown in Figure 7.

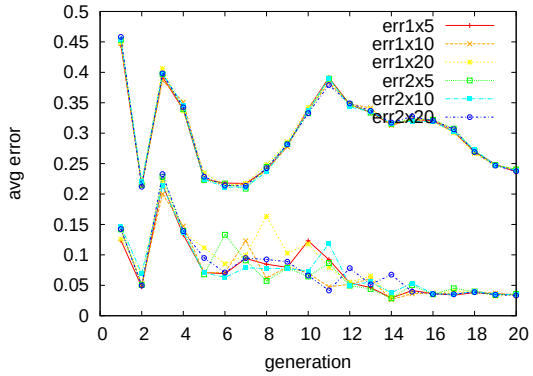



Figure 7: Average error (upper curves) and standard deviation (lower curves) of different types of ANN

[5] 2013-02-12: `http://ls11-www.cs.uni-dortmund.de/rudolph/hypervolume/start`

Even though the huge impact of error (i.e., ups and downs in the graph), the figure show multiple results. First of all, there is almost no difference between different shapes of the ANN. This allows the usage of small and therefore faster configurations. Secondly the estimators show good performance even from the beginning. The initial population (25 candidates, chosen by Latin-hypercube sampling) is sufficient to estimate the first offspring generation. Thirdly, the estimation error in Logarithmic scaling is (with only two exceptions) less than 0.5. We consider this as a good result, because the main role of the estimator is to indicate the magnitude of the objective function for a given candidate.

*5.3.2. Comparing Optimizations with and without Estimator*

Even though the sampling of the search space enabled a great number of repetitions, there are some facts to keep in mind while looking at the subsequent results. Due to the sampling and choosing of the nearest fitting point, multiple points from the domain of the objective functions lead to the same objective values. Points with different parameters but equal objective values are treated as being non-dominating each other. From an engineering point of view this means that there are various different system realizations which are equally good, in terms of the objectives. For the analysis however, we only considered the objective values and not the parameter space. Thus we filtered all multiple occurrences of objective points in every generation.

To give an idea of the general optimization goal for the case study we extracted the Pareto front from the sampled search space. The entire front is depicted as 3D plot in Figure 8. It consists of 83 points, which are shown as black points in the plot. For better presentation we interpolated a grid between the points. The hazard axis shows $log_{10}P(Hazard)$.

For the performance analysis of the optimization we firstly compared the influence of four different ANN configurations on the overall optimization process with an optimization without any estimator. In the Figures 9(a), 9(c) and 9(e) we plotted the Hypervolume metric. The three separate plots show the experiments for a population size of 16, 24 and 32 candidates respectively. In every plot the x-Axis shows the number of generation and the y-Axis shows the average of the Hypervolume indicator. Based on the overall Pareto front extracted from the sampled search space (Figure 8) we choose the reference point $R$ for the Hypervolume indicator as $[Hazard = 0.1, Time = 2.6, Cost = 14]$.

It is well evident that starting from the first offspring generation, the estimation enhanced algorithms are closer to the true front and converge faster than without estimators. Consistent with the analysis of the ANN, there is only little difference between the various shapes of the ANN, i.e., for all three population sizes the Hypervolume metric curves for the ANN enhanced approaches hardly differ from each other.

In addition to the Hypervolume metric we made the same comparison by the number of Pareto optimal points in every generation. The Figures 9(b), 9(d) and 9(f) show the results for a population size of 16, 24 and 32 candidates respectively. As in the plots on the left, the x-Axis shows the number of generations. The y-Axis now shows the number of Pareto optimal points found by

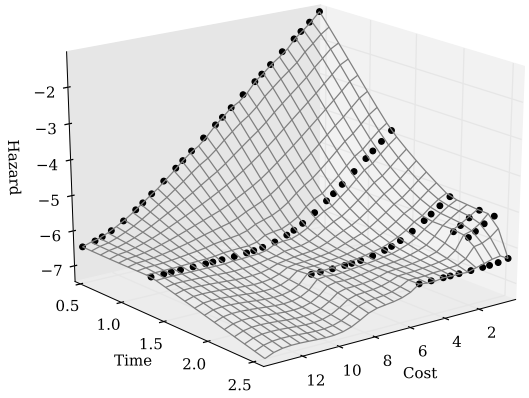


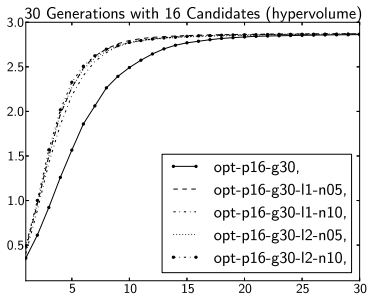

Figure 8: The Pareto Front Interpolated from the Sampled Search Space

the optimization up to the current generation. From the plots it can be seen that the estimation enhanced variants of the algorithm create significantly more Pareto optimal points in every generation than the ones without.

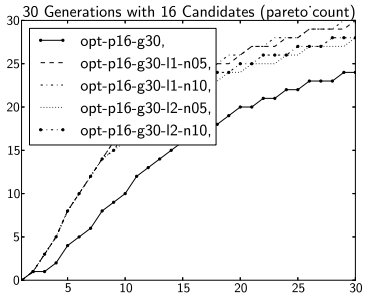
In Figure 9 the x-Axis depicts the generations and the y-Axis the measurement, the number of candidates in every generation was 16 ($a$ and $b$), 24 ($c$ and $d$) and 32 ($e$ and $f$). In summary, these results show that the optimization algorithms in combination with the ANN estimator generate more Pareto optimal solutions in a shorter time than the approach without estimation. Thus, the estimation supported algorithm converges to the Pareto front with less objective function evaluations. This is especially important when considering the great computational costs of the objective functions.

An interesting question is also how the population size influences the optimization results. We thereto plotted the Hypervolume and Pareto count measurements for the population size of 16, 24 and 32 candidates in Figure 10. For better comparability the x-Axis now shows the number of objective function evaluations instead of generations. Due to the fact that the measurements are performed on a generational basis, the values in between are interpolated linearly. The y-axis in Figure 10(a) shows the Hypervolume indicator and in Figure 10(b) it shows the number of Pareto optimal points. All experiments used an artificial neural network with one hidden layer and five neurons in the hidden layer.

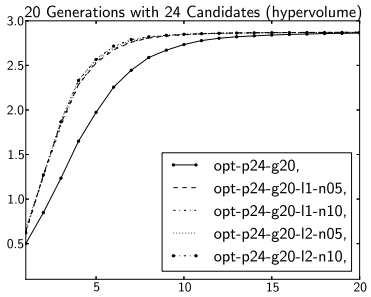
In terms of the Hypervolume metric the experiments with smaller population size show faster convergence towards the reference front. At the same time it appears that the experiment with the smallest population size (i.e., 16 candidates) creates a larger number of Pareto optimal points in the beginning and



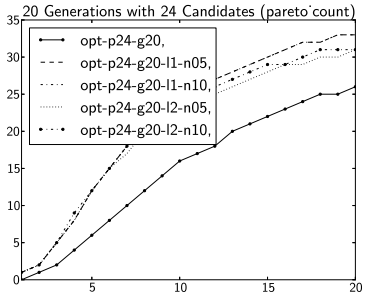

(a) 16 Candidates, Hypervolume Metric (log)



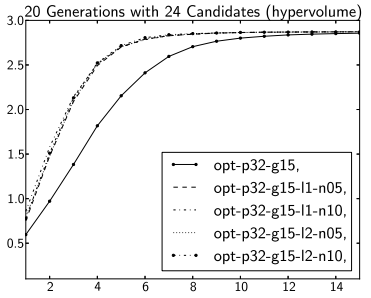

(b) 16 Candidates, Pareto-Count Metric



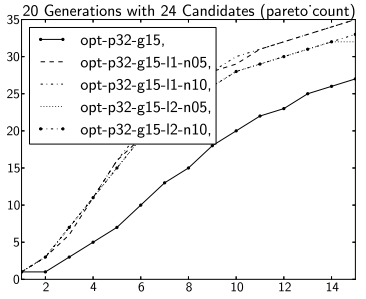

(c) 24 Candidates, Hypervolume Metric (log)




(d) 24 Candidates, Pareto-Count Metric




(e) 32 Candidates, Hypervolume Metric (log)



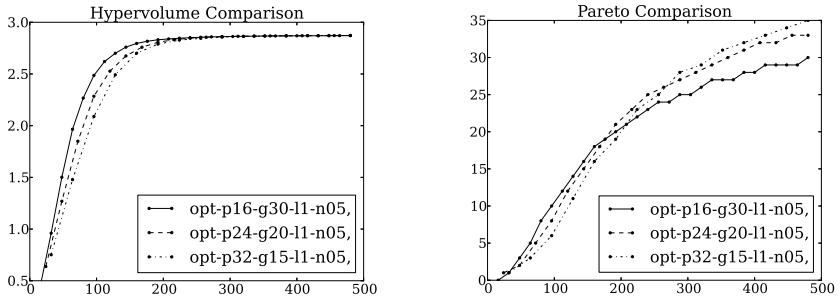

(f) 32 Candidates, Pareto-Count Metric

Figure 9: Hypervolume and Pareto-Count Measurements of the Optimization Experiments

then falls behind the experiments with larger population sizes.

## 6. Related Work

An earlier approach to use hazard probabilities for safety optimization has already been described in [35]. In contrast to the approach based on quan-




(a) Hypervolume Metric (log)

(b) Pareto-Count Metric

Figure 10: Population Sizes vs. Number of Objective Function Evaluations

titative model-based safety analysis, it used analytic mathematical models in an a-posteriori optimization. This approach is computationally more efficient. However, it cannot cope with stochastic dependencies and relies on separate models for quantitative analysis with much coarser quantitative approximations. On the other hand, it might be interesting to use such a model instead of ANN estimation to identify good candidates for evaluation.

Papadopoulos et al. used in [37] the Hip-Hops [36] methodology as basis for optimization of safety critical systems. Hip-Hops is a structural approach to safety analysis, where components with known properties are combined into a larger system model and a failure propagation model describes the reaction of the system to the occurrence of failures. Such a structural description is then used to evolve better system designs with similar properties wrt. functionality, but better failure tolerance and/or lower costs. For this optimization a variant of NSGA-II is used.

The advantage of our model-based approach is the accompanying increase in accuracy. A disadvantage is that the whole system must be analysed, and therefore it may suffer from the state-explosion problem. This is less severe for an approach as [37]. A combination of the two approaches would be interesting, e.g., using the Hip-Hop approach for the structural description of the system, but analyzing the single components with the more accurate model-based safety analysis. The optimization could then be applied on either structural level – exchanging equivalent components – or the parameter level of the components. This could be an interesting compromise between accuracy and scalability.

There are also some approaches for finding optimized system variants which are not necessarily focused on safety: Islam et al. describe an interesting approach in [24]. They apply a mapping from an abstract specification of a system to a concrete implementation which is optimized under different aspects. It is specifically aimed at the development of real-time fault-tolerant systems, where software functionality is mapped onto communicating hardware. They then use general constraint solver and optimization software like CPLEX to find the best allocation of tasks. Although similar, the approach is less general and very fo-

cused on task allocation. It supports only scalar objective functions, i.e., for multi-objective optimization it would be necessary to apply scalarization techniques.

There exist some approaches for design space exploration (DSE). Hegedüs et al. [22] for example describe an approach based on a UML model description where system states are described as instances of a meta-model. The model allows an abstract representation of the requirements and uses guidance (either heuristic or manual) to find best realizations of system variants. The realizations are derived using graph transformations and the objectives – or goals in this case – are described as graph patterns. Knorreck et al. describe in [25] an approach for formal verification of real-time properties of a UML profile which allows for real-time requirements specification. For the formal analysis, the model is transformed into the input language of the UPPAAL real-time model-checker or the process algebra language LOTOS. The tool then creates different realizations of the system and uses the CADP toolbox [13] to generate and reduce the state space to visible actions. The objective is to find the realization with the least visible (and time-consuming) actions which fulfills the required real-time properties.

These two approaches have the advantage of using graphical model specifications in UML. Currently, they do provide neither probabilistic safety analysis nor the possibility to specify probabilities in the models. On the other hand, the integration of our optimization approach for large probabilistic models into a framework with visual modeling capabilities would be a real advantage and would facilitate its application.

Aleti et al. [1] present an Eclipse based tool for multi-objective optimization of AADL based architecture models. In their publication the authors concentrate on component deployment problems, but their tool seems to be flexible enough to integrate more sophisticated objectives. The tool provides a framework for multiple optimization algorithms and various objective functions. Nevertheless they do not provide any technique to deal with increasing computational costs of analyzing the underlying Markov chain models.

For comparison we summarized the aforementioned approaches in Table 2. There are two approaches ([35], [37]) which strictly focus on safety analysis, i.e., they both rely on either fault tree analysis or failure mode and effects analysis. Two other approaches ([24], [22]) use UML like model semantics. All approaches focus on system structure rather than functional aspects.

Our approach is the only one to use Markov decision processes as model semantics, allowing for the analysis of functional and structural aspects. The usage of such powerful specification formalism clearly comes at the cost of increased computational efforts. We successfully decreased the computation-time by introducing an artificial neuronal network based estimation algorithm for the expensive objective functions.

Our approach and the one presented by Hegedüs et al. [22] heavily differ in the way how the parameters are specified. We directly specify all possible realizations of a system by stating specification families. The values of parameters are used to select specific members of the specification family. In contrast

| Approach | Optimization Algorithm | Model Semantics | Objectives |
|---|---|---|---|
| O-SAML | NSGA-II | MDP | pCTL, CTL, Java |
| [35] | Various Single-Objective | Extended FTA | Cost |
| [37] | NSGA-II | IF-FMEA Annotation | Dependability, Cost |
| [24] | Linear Programming | UML and Custom | Dependability Realtime |
| [22] | Exploratory Search | Custom UML-Based | Bounds on number of matches of graph pattern |
| [1] | Various Evolutionary | AADL/UML | Reliability Performance Others |

Table 2: Comparison of related approaches

to that Hegedüs et al. specify parameters in an inductive way. A set of modification rules which are applicable on an initial model is used to create every possible specification variant. This implies that the search space is difficult to survey. However it enables the finding of specification variants which nobody thought of during specification time.

Looking at the Model-Semantics column in Table 2 our approach is based on the most accurate and sound semantics. While being rather restrictive for modeling, the Markov decision process is formally defined and, in contrast to most UML based notations, leaves no freedom for ambiguity.

Besides the directly optimization related approaches, Grunske proposes a framework for early quality prediction of component-based systems [14]. It is primarily aimed at systems where reliability, safety, availability and security are important, but also allows consideration of performance of the system. Although it mainly considers software systems, the framework could be extended to also include software-intensive safety critical systems and additional non-functional requirements like costs as described in this paper. This would allow the described model-based multi-objective safety optimization to be integrated into the larger framework proposed in [14].

## 7. Conclusion

We presented an approach for multi-objective optimization of large probabilistic models. Besides generic objective functions (i.e., cost or performance) expressible in Java, we particularly facilitate the usage of probabilistic model checking as objective function. This allows the optimization of safety related

properties. We increase the efficiency of the genetic algorithm based approach by the introduction of estimation techniques for the most costly objective functions.

The available API allows for the specification of arbitrary objective functions in the Java language. Interfaces allow for the easy integration of different verification engines and also for experiments using different estimation methods and strategies. We support PCTL based objectives and artificial neural network based estimation strategies out of the box.

We presented several experiments of the application of our approach to a safety-critical real-world case study from the railway domain. The results showed that the application of an artificial neural network increases the performance of the optimization algorithm and thus reduces the computational effort. This is a bid advantage, considering that a simple optimization took more than 4 days of calculation time on a modern computer.

Due to the stochastic behavior of evolutionary algorithms and of artificial neuronal networks, we repeated every experiment 500 times and statistically analyzed the results. To cope with the computational complexity of the optimization problem we sampled the whole search space of the case study and created a database of objective-function values for every sample point. The experiments itself then used the pre-calculated data from the objective database. Thus we were able to run multiple experiments in a feasible amount of time.

In conjunction with the $S^3E$ tool, an Eclipse and SAML based system specification and analysis tool, the model generation and optimization is convenient and highly automated.

**Acknowledgement:**

Simon Struck is funded by the German Research Foundation (DFG) within the Probabilistic Models for Safety Analysis (ProMoSA) project.

[1] Aleti, A., Bjornander, S., Grunske, L., Meedeniya, I., 2009. Archeopterix: An extendable tool for architecture optimization of aadl models. In: Model-Based Methodologies for Pervasive and Embedded Software, 2009. MOMPES'09. ICSE Workshop on. IEEE, pp. 61–71.

[2] Aljazzar, H., Fischer, M., Grunske, L., Kuntz, M., Leitner-Fischer, F., Leue, S., 2009. Safety analysis of an airbag system using probabilistic fmea and probabilistic counterexamples. In: Proceedings of the 2009 Sixth International Conference on the Quantitative Evaluation of Systems (QEST 2009). IEEE Computer Society, Washington, DC, USA, pp. 299–308.

[3] Bianco, A., de Alfaro, L., 1995. Model checking of probabalistic and nondeterministic systems. In: Foundations of Software Technology and Theoretical Computer Science. pp. 499–513.

[4] Booker, L. B., Goldberg, D. E., Holland, J. H., 1989. Classifier systems and genetic algorithms. Artificial Intelligence 40 (1-3), 235–282.

[5] Bozzano, M., Cimatti, A., Katoen, J.-P., Nguyen, V. Y., Noll, T., Roveri, M., 2010. Safety, dependability, and performance analysis of extended AADL models. The Computer Journal.

[6] Bozzano, M., Villafiorita, A., 2003. Improving system reliability via model checking: the FSAP/NuSMV-SA safety analysis platform. In: Proceedings of the $22^{nd}$ International Conference on Computer Safety, Reliability and Security (SAFECOMP 2003). Springer, pp. 49–62.

[7] Bozzano, M., et al., 2003. ESACS: An integrated methodology for design and safety analysis of complex systems. In: Proceedings of European Safety and Reliability Conference (ESREL 2003). Balkema publisher, Maastricht, The Netherlands, pp. 237–245.

[8] Clarke, E., Grumberg, O., Peled, D., 2000. Model Checking. MIT Press.

[9] de Alfaro, L., Faella, M., Henzinger, T. A., Majumdar, R., Stoelinga, M., 2005. Model checking discounted temporal properties. Theoretical Computer Science 345, 139–170.

[10] Deb, K., Agrawal, R., 1994. Simulated binary crossover for continuous search space. Complex Systems 1 (9), 115–148.

[11] Deb, K., Pratap, A., Agarwal, S., Meyarivan, T., 2002. A fast and elitist multi-objective genetic algorithm: NSGA-II. IEEE Transaction on Evolutionary Computation, 181–197.

[12] Distefano, S., Puliafito, A., 2007. Dft and drbd in computing systems dependability analysis. In: SAFECOMP. pp. 423–429.

[13] Garavel, H., Lang, F., Mateescu, R., Serwe, W., 2011. CADP 2010: A Toolbox for the Construction and Analysis of Distributed Processes. In: Proc. of TACAS'11. Vol. 6605 of LNCS. Springer.

[14] Grunske, L., 2007. Early quality prediction of component-based systems - a generic framework. Journal of Systems and Software 80 (5), 678 – 686, component-Based Software Engineering of Trustworthy Embedded Systems.

[15] Güdemann, M., 2011. Qualitative and Quantitative Formal Model-Based Safety Analysis. Ph.D. thesis, Otto-von-Guericke-Universitä Magdeburg. URL `http://nbn-resolving.de/urn:nbn:de:gbv:ma9:1-385`

[16] Güdemann, M., Ortmeier, F., 2010. A framework for qualitative and quantitative model-based safety analysis. In: Proceedings of the $12^{th}$ High Assurance System Engineering Symposium (HASE 2010). pp. 132–141.

[17] Güdemann, M., Ortmeier, F., 2010. Probabilistic model-based safety analysis. In: Proceedings of the $8^{th}$ Workshop on Quantitative Aspects of Programming Languages (QAPL 2010). EPTCS, pp. 114–128.

[18] Güdemann, M., Ortmeier, F., 2011. Model-Based Multi-Objective Safety Optimization. In: Proceedings of the $30^{th}$ International Conference on Computer Safety, Reliability and Security (SAFECOMP 2011). Springer LNCS, pp. 423–436.

[19] Güdemann, M., Ortmeier, F., 2011. Towards Model-driven Safety Analysis. In: Proceedings of the 3rd international Workshop on Dependable Control of Discrete Systems (DCDS 2011). IEEE, pp. 53–58.

[20] Hansson, H., Jonsson, B., 1994. A logic for reasoning about time and reliability. Formal Aspects of Computing 6, 102–111.

[21] Hassoun, M. H., 1995. Fundamentals of artificial neural networks. MIT Press.

[22] Hegedus, A., Horvath, A., Rath, I., Varro, D., 2011. A model-driven framework for guided design space exploration. In: Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering. ASE '11. IEEE Computer Society, Washington, DC, USA, pp. 173–182.

[23] Igel, C., Hüsken, M., 2000. Improving the rprop learning algorithm. In: Proceedings of the second international ICSC symposium on neural computation (NC 2000). Citeseer, pp. 115–121.

[24] Islam, S., Suri, N., Balogh, A., Csertan, G., Pataricza, A., 2009. An optimization based design for integrated dependable real-time embedded systems. Design Automation for Embedded Systems 13, 245–285.

[25] Knorreck, D., Apvrille, L., Pacalet, R., 2012. Formal system-level design space exploration. Concurrency and Computation: Practice and Experience, n/a–n/a.

[26] Kwiatkowska, M., Norman, G., Parker, D., 2002. Probabilistic symbolic model checking with PRISM: A hybrid approach. In: Proceedings of the $8^{th}$ International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2002). Vol. 2280 of LNCS. Springer.

[27] Kwiatkowska, M., Norman, G., Parker, D., 2011. Prism 4.0: Verification of probabilistic real-time systems. In: Computer Aided Verification. Springer, pp. 585–591.

[28] Lipaczewski, M., Struck, S., Ortmeier, F., 2012. Saml goes eclipse - combining model-based safety analysis and high-level editor support. In: Proceedings of the 2nd International Workshop on Developing Tools as Plug-Ins (TOPI). IEEE, pp. 67–72.

[29] Meedeniya, I., Buhnova, B., Aleti, A., Grunske, L., 2010. Architecture-driven reliability and energy optimization for complex embedded systems. Research into Practice–Reality and Gaps, 52–67.

[30] Nissen, S., 2003. Implementation of a fast artificial neural network library (fann). Tech. rep., Department of Computer Science University of Copenhagen (DIKU), http://fann.sf.net.

[31] Norman, G., Parker, D., Kwiatkowska, M., accessed: June 2010. The PRISM language - semantics.
URL `http://www.prismmodelchecker.org/doc/semantics.pdf`

[32] Ortmeier, F., Güdemann, M., Reif, W., 2007. Formal failure models. In: Proceedings of the $1^{st}$ IFAC Workshop on Dependable Control of Discrete Systems (DCDS 2007). Elsevier.

[33] Ortmeier, F., Guedemann, M., Lipaczewski, M., 2011. Practical experiences in model-based safety analysis. In: proceedings: International Workshop on Digital Engineering. ACM Proceedings, pp. 31–38.

[34] Ortmeier, F., Reif, W., Schellhorn, G., 2005. Formal safety analysis of a radio-based railroad crossing using deductive cause-consequence analysis (DCCA). In: Proceedings of 5th European Dependable Computing Conference (EDCC 2005). Vol. 3463 of LNCS. Springer.

[35] Ortmeier, F., Schellhorn, G., Reif, W., 2004. Safety optimization of a radio-based railroad crossing. In: Schnieder, E., Tarnai, G. (Eds.), Proceedings of Formal Methods for Automation and Safety in Railway and Automotive Systems (FORMS / FORMAT 2004).

[36] Papadopoulos, Y., McDermid, J., 1991. Hierarchically performed hazard origin and propagation studies. In: Computer Safety, Reliability and Security.

[37] Papadopoulos, Y., Walker, M., Parker, D., Rüde, E., Hamann, R., Uhlig, A., Grätz, U., Lien, R., 2010. Engineering failure analysis and design optimisation with HiP-HOPS. Engineering Failure Analysis.

[38] Parr, T., 2007. The Definitive ANTLR Reference: Building Domain-Specific Languages. Pragmatic Programmers. Pragmatic Bookshelf.

[39] Åkerlund, O., Bieber, P., Boede, E., Bozzano, M., Bretschneider, M., Castel, C., Cavallo, A., Cifaldi, M., Gauthier, J., Griffault, A., Lisagor, O., Luedtke, A., Metge, S., Papadopoulos, C., Peikenkamp, T., Sagaspe, L., Seguin, C., Trivedi, H., Valacca, L., 2006. ISAAC, a framework for integrated safety analysis of functional, geometrical and human aspects. In: Proceedings of 2nd European Congress on Embedded Real Time Software (ERTS 2006).

[40] Reif, W., Schellhorn, G., Thums, A., June 2000. Safety analysis of a radio-based crossing control system using formal methods. In: Schnieder, E., Becker, U. (Eds.), Proceedings of the $9^{th}$ IFAC Symposium Control in Transportation Systems (CTS 2000).

[41] Walker, M., Bottaci, L., Papadopoulos, Y., 2007. Compositional Temporal Fault Tree Analysis. In: Proceedings of the $26^{th}$ International Conference on Computer Safety, Reliability and Security (SAFECOMP 2007).

[42] Zitzler, E., Thiele, L., 1998. Multiobjective Optimization Using Evolutionary Algorithms – A Comparative Case Study. In: Parallel Problem Solving from Nature—PPSN V. Springer, pp. 292–301.