TECHNISCHE UNIVERSITÄT DRESDEN

Dep. of Computer Science Institute for System Architecture, Database Technology Group

Master Thesis

ML-SUPPORTED DESIGN METHODS FOR DETERMINING PERMISSIBLE INPUT VARIABLES FOR CRASH SIMULATION USING THE EXAMPLE OF FEM

Julian Haluska

Matr.-Nr.: 4671056

Supervised by:

Prof. Dr.-Ing. Wolfgang Lehner

and:

Dr.-Ing. Maik Thiele

Submitted on March, 13th, 2020

CONFIRMATION

I confirm that I independently prepared the thesis and that I used only the references and auxiliary means indicated in the thesis.

Dresden, March, 13th, 2020

ABSTRACT

In the field of engineering the design process requires that component designs satisfy specific system design goals. These designs are called good designs. It is desirable that components can be designed independently and robust with respect to variation. This problem is equivalent to determining the largest box in the subspace of good designs. Such a box is a cartesian product of permissible design parameter intervals. Typically, the classification into good and bad designs is computed by numerical methods (e.g. FEM) and in few cases given by real experiments (e.g. crash tests in automotive design). Hence, the design space is approximated by a finite number of points. This work develops a software prototype which determines permissible intervals on this approximated design space. Essential steps in the pipeline are outlier detection, data augmentation with machine learning, clustering and box maximization. For box maximization an exact as well as two heuristic algorithms are developed. The box maximization algorithms are evaluated on synthetic data in order to have a benchmark for comparison. It is shown that the runtime grows with the amount of designs and dimensions. A heuristic algorithm based on the meta-heuristic simulated annealing mostly finds good solutions on the tested cases. Furthermore, the prototype is evaluated on two real automotive crash datasets. The end-to-end evaluation shows that the prototype is capable of removing outliers, augmenting the design space and providing permissible intervals of design parameters for problems with arbitrary dimensionality.

CONTENTS

Acknowledgments **11**

1 Introduction **13**

1.1 Nonlinearity in the Design Process 13

1.2 Decoupling Design Parameters 14

1.3 Practical Limitations 15

1.4 Thesis Structure 15

2 Concept Development **17**

2.1 Problem Statement 17

2.2 Data Preprocessing 19

2.2.1 Outlier Detection 20

2.2.2 Data Augmentation 20

2.3 Determining Permissible Intervals 21

2.3.1 Related Work 21

2.3.2 Multiple Solution Boxes 22

2.4 Concept 23

3 Methods **27**

3.1 Outlier Detection 27

3.1.1 Proximity-Based Methods 28

3.1.2 Isolation Forest 29

3.2 Data Augmentation 30

3.2.1 Model Generation 30

3.2.2 Sampling Methods 34

3.3 Determining Permissible Intervals 35

3.3.1 Clustering 35

3.3.2 Box Maximization 36

4 Implementation **39**

4.1 Implementation Specifics 39

4.1.1 Data Preprocessing 39

4.1.2 Determining Permissible Intervals 40

4.2 User Workflow 42

5 Component-Wise Evaluation **45**

5.1 Outlier Detection 45

5.1.1 K Nearest Neighbour 46

5.1.2 DBSCAN 46

5.1.3 Isolation Forest 47

5.1.4 Summary 48

5.2 Clustering 49

5.3 Box Maximization 49

5.3.1 Linear Problems 51

5.3.2 Nonlinear Problems 59

5.3.3 Runtime Difference between Grid and Sobol Sampling with Exact Max Box 62

5.3.4 Summary 63

6 End-to-End Evaluation **65**

6.1 SCALE dataset 65

6.1.1 Outlier Detection and Model Quality 65

6.1.2 Data Augmentation 66

6.1.3 Clustering 67

6.1.4 Box Maximization 67

6.2 US-NCAP Dataset 69

6.2.1 Outlier Detection and Model Quality 69

6.2.2 Data Augmentation 70

6.2.3 Clustering 70

6.2.4 Box Maximization 70

6.3 Summary 72

7 Discussion **73**

8 Conclusion **75**

A Appendix **77**

A.1 Real Datasets 77

A.1.1 SCALE Dataset 77

A.1.2 US-NCAP Dataset 77

A.2 Synthetic Data 77

A.2.1 Functions for Data Generation 79

A.2.2 Generated Datasets 80

A.3 Proofs 80

ACKNOWLEDGMENTS

This thesis was written and elaborated at the Database Systems group of the TU Dresden in cooperation with the SCALE GmbH.

I am very grateful to Prof. Dr.-Ing Wolfgang Lehner for the supervision of this thesis. I want to thank my academic advisor at the Database systems group, Dr.-Ing Maik Thiele, for the continual help and feedback.

Furthermore, I want to thank Dr. Martin Liebscher at SCALE for giving me the opportunity to write about a practical use case of the automotive industry and the expressed trust in my work. Likewise, I want to thank my advisor at SCALE, Dr. Reinhard Stahn, for the many ideas, suggestions, discussions and mathematical explanations. They contributed greatly to the presented work.

1 INTRODUCTION

The design of structural components in automotive manufacturing is an iterative process consisting of volatile and rough requirements [32]. To decrease production costs, engineers make use of computer-aided methods (abbreaviated CAx, Computer-Aided x) in the early phase of product development. The CAx process chain itself is composed of two main tasks: computer-aided design (CAD) and computer-aided engineering (CAE) ([32], Section 2.3). Steber states that 'the objective of CAE is to determine the functional properties of components with a physical substitue model as accurate as possible'. For that reason, forces on objects and resulting deformations are computed using numerical methods like the finite element method (FEM). An example is crash simulation where the deformation of a vehicle in an accident is computed. This evades the execution of an expensive crash test with a real vehicle.
Part of the CAx process is the design of structural components in a way to meet requirements for the whole vehicle. Those requirements are called *system design goals*. At the same time, engineers want to optimize their structural components for different *component design goals*, such as weight and cost [33]. In the case of crash testing for example the system design goal could be that the deformation of the vehicle stays in an acceptable range. A component design goal would be to optimize a suspension arm in a way to reduce production costs, while still satisfying the overall system design goals of the vehicle. During the design process, engineers need to have these design goals in mind.

1.1 NONLINEARITY IN THE DESIGN PROCESS

Designs can be understood as a set of *design parameters* [33]. A design can be evaluated by an arbitrary *function* to get a *system response*. In terms of our crash simulation example, the function is the crash simulation, a design parameter can be the belt force level on a dummy's seatbelt and the system response can be the force on the neck of a dummy. In this case engineers would try to design the vehicle in a way that the force on the neck of the dummy is not too high in order to ensure the safety of the passenger. Constraints on such a force can be seen as a *design goal*, which is either satisfied or not. This divides the designs in *good* or *bad* designs. Due to the complex, nonlinear nature of the modeled system (e.g. crash simulation), the shape of regions with good

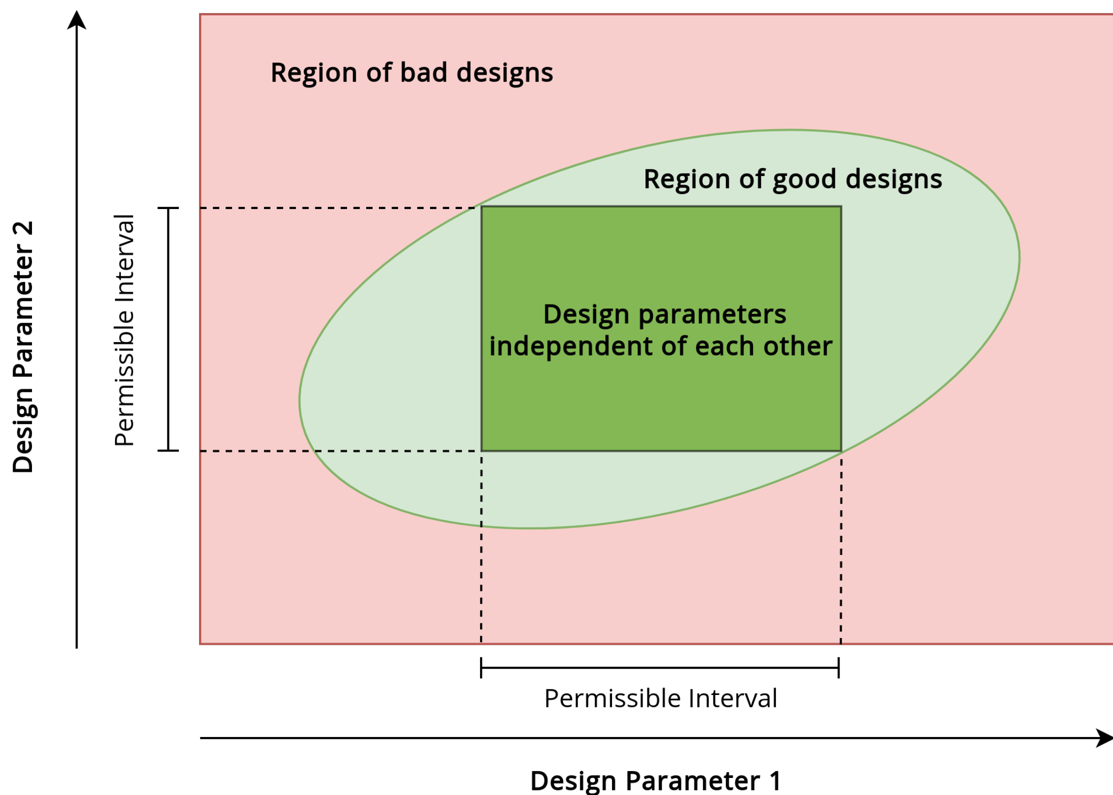


Figure 1.1: A two-dimensional design space with good and bad designs. An axis-parallel box of good designs is equivalent to a permissible interval for each design parameter. This idea generalizes into multidimensional design spaces.

and bad designs can be arbitrary.

When optimizing designs, engineers commonly have to choose specific values for those design parameters. The selection of these values is difficult because the modeled system is complex: parameters interact with one another and behave nonlinear. A change in one parameter can cause a completeley different system response; while a change of another parameter might not affect the system response at all. The modeled systems are more sensitive to some parameters, to others less.

With trial and error and over time, engineers gain an intuitive understanding of the influence of their parameter selection on the system response. With this knowledge it is possible to make educated guesses on the basis of sparse data. However, this process is very time-consuming and requires expert knowledge. Therefore, there exists a need to support engineers in the process of selecting permissible design parameters; meaning design parameters which result in good designs.

1.2 DECOUPLING DESIGN PARAMETERS

From the perspective of engineers it would be easier to choose values for design parameters freely from intervals, where they can be confident that the system response will still meet their design goal. We refer to these intervals as *permissible intervals*.

In general, as the space of good designs can be of arbitrary shape, it is not possible to do that.

In most cases the system response and therefore, the distinction between good and bad designs inherently depends on the **combination** of design parameters. To be able to choose a value for a parameter independently of other parameters, the parameter selection has to be decoupled from one another. In a spatial sense this means that the objective is to find multidimensional axis-parallel boxes of good designs. Figure 1.1 shows this visually.
The main goal of the software prototype developed in this thesis is to find axis-parallel boxes of good designs which are equivalent to permissible intervals for each design parameter. Since there is an infinite amount of possible boxes for regions of good designs, the goal is to find a preferably large box, to provide engineers with large intervals to choose from. By finding those boxes, we can reduce the complexity of choosing suitable design parameters for the engineer. The key idea is to simplify the complex problem of design parameter selection to a linear problem.

1.3 PRACTICAL LIMITATIONS

In order to provide engineers with a solution to the proposed problem, three practical limitations have to be taken into account:

1. The solution provided in this thesis assumes that designs are provided as data points without any knowledge about the process which created it. Therefore, the function of the underlying process is neither known analytically nor numerically.
2. As data in practice is often assembled from different sources with the possibility of human error, there has to be a preprocessing step to find outliers. They need to be removed.
3. It can not be assumed that the provided data fills the design space evenly. There may be gaps for specific design parameter combinations.

1.4 THESIS STRUCTURE

The goal of this thesis is to implement a software prototype which determines independent multidimensional intervals of permissible design parameters for a given dataset of designs consisting of design parameters and system responses.

At the beginning of Chapter 2 a problem statement defines the problem space theoretically. On the basis of related work a concept for a software solution is created which considers the practical limitations stated in Section Section 1.3. As a result, the proposed solution needs to leverage specific technologies, such as outlier detection, model generation, clustering and optimization. Therefore, we revisit these methods in Chapter 3. Chapter 4 explains implementation specifics and the user workflow of the prototype. In Chapter 5 the distinct components of the prototype are evaluated on the basis of synthetic data. In Chapter 6 the whole prototype is evaluated on the basis of real datasets. In Chapter 7 the findings of the evaluation and its implications are discussed. Eventually, we sum up the work and give a conclusion.

2 CONCEPT DEVELOPMENT

In the Introduction we sketched the engineer's problem of optimizing good designs in a way that they still meet their design goals. For this we proposed a solution to simplify the process of design parameter selection by providing intervals of independent design parameters.
In this chapter we develop a comprehensive concept to solve this problem. At first the problem space is defined theoretically in a problem statement. Then, a data preprocessing step is introduced which addresses practical problems with the data. Afterwards related work on determining permissible intervals is examined and used as a basis to create an concept for a solution. In the end the solution is summarized.

2.1 PROBLEM STATEMENT

Zimmermann et al. [33] define *designs* or *design points* as vectors $\mathbf{x} = (x_1, \dots, x_p), x_i \in \mathbb{R}, p \in \mathbb{Z}$ where p is the dimensionality of the vector. All possible designs span *design space* Ω_{ds}. Designs can be evaluated by a *performance function* $f(x)$, which is not known analytically and, in general, can be numerically calculated. It calculates a *system response* $\mathbf{z}$:

$$z = f(x) \tag{2.1}$$

No statement is made about the domain of $\mathbf{z}$. In our case, we assume $\mathbf{z}$ to be multidimensional, i.e. $\mathbf{z} = (z_1, \dots, z_q), z_i \in \mathbb{R}, q \in \mathbb{Z}$ with q being the dimensionality of the system response. Furthermore, a *threshold value* f_c on the system response is defined. For our purpose, we define $f_c = [f_c^l, f_c^u]$ where the first vector is the lower constraint and the second vector is the upper constraint on the system response. Thus, we get the equation 2.2:

$$f_c^l \leqslant f(x) \leqslant f_c^u \tag{2.2}$$

It is defined that a *good* design satisfies the equation 2.2. Otherwise designs are designated as *bad*

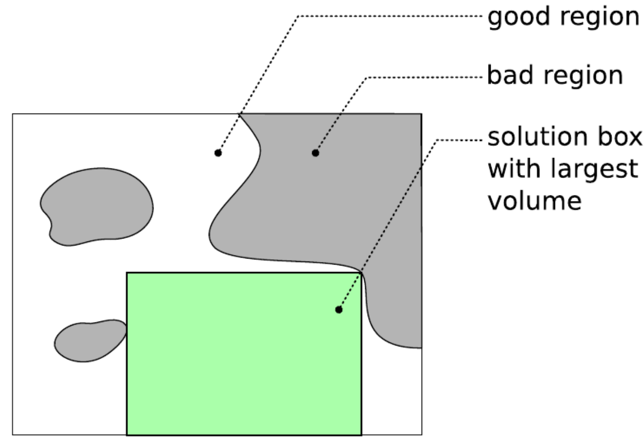


Figure 2.1: Solution space consisting of design points divided into good and bad regions with the largest possible solution box [33].

designs. The *solution space* consisting of the system responses of designs is divided in *good regions* and *bad regions*. In Figure 2.1 this is shown with an example in two dimensions. Furthermore, the largest possible box of good designs is shown.
If there is only one design parameter dimension, the desired solution box is a simple interval with two numbers. With two input dimension, we need two intervals - one for each dimension - in which all the points predict good designs. This corresponds to a rectangle as depicted in Figure 2.1. In three dimensions we need three intervals and get an rectangular cuboid, and so on. This idea generalizes into multidimensional space. Mathematicians call these constructs *hyperrectangles* or *n-orthotopes*. Eckstein et al. name the construct *box*. The important property of such a box of good designs is that the design parameters are independent of each other. Thus, Zimmermann et. al. define a *solution box* as

$$\Omega = I_1 \times I_2 \times \ldots \times I_p \tag{2.3}$$

where $I_i = [x_i^l, x_i^u], i \in [1, p]$. Note, that a p-dimensional box can be defined by two p-dimensional vectors. The superscript l refers to the lower bound of the box and u to the upper bound.

To decide whether a solution box is better than another one, Zimmerman et al. propose the volume as a metric. Additionally, they define the volume ratios of good and bad regions inside a solution box as *fraction of good designs*. However, in our case we want the solution boxes to purely consist out of good designs, i.e. we do not allow any bad designs in our solution boxes. Therefore, we propose a conservative metric. Our solution boxes should contain at least one design point, i.e. boxes that do not contain any design will have a score of minus infinity. Also we want to punish solution boxes which contain bad designs by a negative score which equals to the amount of bad designs. Hence, we define a *box fitness* as

$$\mu(\Omega) = \begin{cases} -\infty & \text{, if } b = 0, g = 0 \\ -b & \text{, if } b > 0 \\ 100(x_1^u - x_1^l)(x_2^u - x_2^l)\ldots(x_p^u - x_p^l) & \text{else} \end{cases} \tag{2.4}$$

where b is the amount of bad and g is the amount of good designs. The volume in the else-case was multiplied by 100 to have a percentage value.
The box fitness is the metric with which we will compare solution boxes. With the box fitness and based on the definition of Zimmermann et. al we get the optimization problem:

$$f_c^l \leqslant f(x) \leqslant f_c^u, \ for \ all \ \mathbf{x} \subseteq \Omega \tag{2.5}$$

$$\mu(\Omega) \rightarrow max \tag{2.6}$$

In this thesis, we refer to this optimization process as *box maximization*.

2.2 DATA PREPROCESSING

In order to provide the process of determining permissible intervals with useful data, we develop means to address the shortcomings of the provided data. The practical limitations of the data were stated in Section 1.3. From that, two problems for data preparation arise:

1. Data could contain outliers.
2. Designs can be either sparse or unevenly distributed in the design space.

The first problem is addressed by outlier detection. The second problem results in a need to perform data augmentation. The resulting data preprocessing pipeline is shown in Figure 2.2).

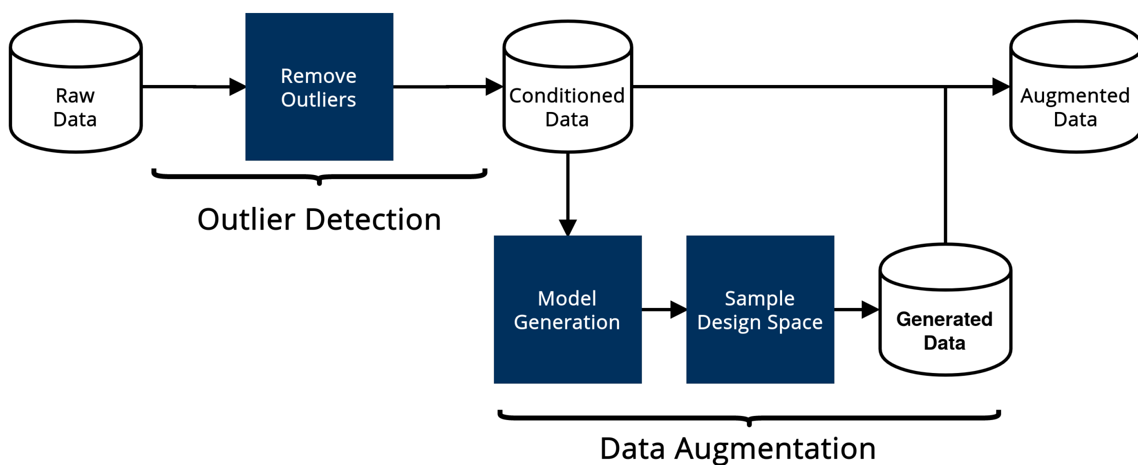


Figure 2.2: Data preprocessing pipeline.

2.2.1 Outlier Detection

Naturally, data points which were generated by a different process than the tested process can skew the results of any analysis performed on the dataset. Since we can not guarantee that the provided data is free of outliers, there is the need to provide the engineer with methods to find and remove outliers. The basic way to detect outliers is to plot the dataset in a scatterplot matrix. However, in large datasets comprised of thousands of samples and many columns, outliers can not be detected by the human eye easily. Also, scatterplot matrices tend to grow fast and become cluttering. Therefore, we need to support the engineer in the process of outlier detection. An additional reason to remove outliers is the fact that models will be generated based on the data. The prediction accuracy can be negatively affected by outliers. Therefore, it is necessary to eliminate outliers as a source of error.
After removing outliers from the initial dataset the data is conditioned for the next preprocessing step.

2.2.2 Data Augmentation

The second step of the data preprocessing pipeline is concerned with the sparse and uneven distribution of data in the design space. Sparse datasets are common because it is not possible to test combinations of design parameters exhaustively. Simulations can take days to successfully execute and real crashtests are even more time- and resource consuming. As a result datasets often contain large regions without designs. The behaviour of designs in those areas is uncertain. This leads to solution boxes with uncertain validity. An example is visualized in Figure 2.3.
To improve the validity of solution boxes, the design space needs to be filled more evenly. Since the underlying model which generated the data is neither known analytically nor numerically, the only way to provide information about regions without designs is to interpolate or extrapolate from the existing dataset. Therefore, a model of the data is needed with which the design space can be filled. Such models can be created by a machine learning approach. With a model it is possible to predict designs with arbitrary design parameter combinations. Hence, a model generation step is proposed.
While creating new designs the question arises how to decide where in the design space to create new designs. Since the amount of sampling points has to be fixed, points should be distributed evenly in the design space in order to have the most information about designs. Therefore, there exists a need for sampling methods which distribute designs evenly in the design space.
Therefore, in summary, data augmentation consists of two steps:

1. Generating a model of the data (Model Generation)

2. Filling the design space by sampling evenly (Design Space Sampling)

After creating new designs, this additional data is added to the conditioned dataset of the preceding step. This is the second step in Figure 2.2.

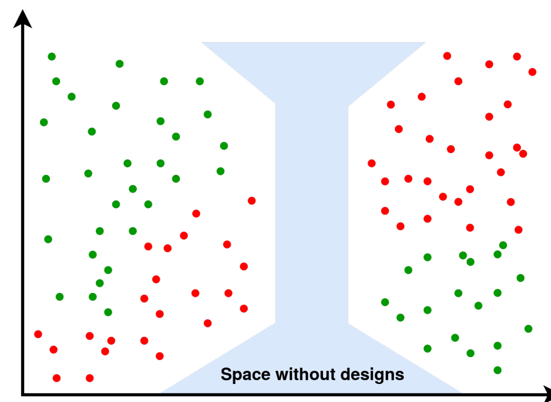


Figure 2.3: In the provided data certain combinations of design paramers can be missing. As a result, there are visible gaps (blue) in the design space . Solution boxes comprising this space can not make any assumption about the validity of a design from these areas.

2.3 DETERMINING PERMISSIBLE INTERVALS

After the data is conditioned and augmented, it can be used to determine permissible intervals. To explore the problem further, at first we review related work on the topic of determining permissible intervals.

2.3.1 Related Work

Harbrecht et al. determine polygons of good regions in solution spaces [12]. However, polygons do not fulfill the need of design parameters to be independent of eacht other. Zimmermann et al. on the other hand determine the solution boxes in their paper *Computing Solution Spaces For Robust Design* [33]. A stochastic algorithm is proposed which is evaluated in a subsequent paper *On the computation of solution spaces in high dimensions* [9]. The algorithm consists of two phases and two modification steps. The two modification steps, denoted as A and B, have different tasks. Modification step A removes bad designs from a box until only good designs are present in a box. Modification step B tries to enlarge the box while allowing for a certain small amount of bad designs to be part of the box. In the first phase of the algorithm the box is enlarged by modification step B first. Then, new sampling points are generated by Monte Carlo sampling inside that box. Then the bad designs are removed by the modification step A. The first phase is repeated as long as the solution box is changing. In the second phase the box is shrinked by applying modification step A only. This process of iteratively increasing the size of a solution box is visualized in Figure 2.4.
The runtime differs depending on the runtime of the Monte Carlo sampling which creates new design points on-the-fly during execution. Zimmermann et al. state that the runtime is in $O(vN^2p)$, where v is the number of iteration steps, N is the amount of Monte Carlo sampling points generated in each iteration and p is the number of dimensions. This runtime only holds if the computation of f - the function to generate data points with - is inexpensive, i.e. constant. In our context f is unknown. Nevertheless, it is possible to generate a model of the provided dataset which approximates the function f.

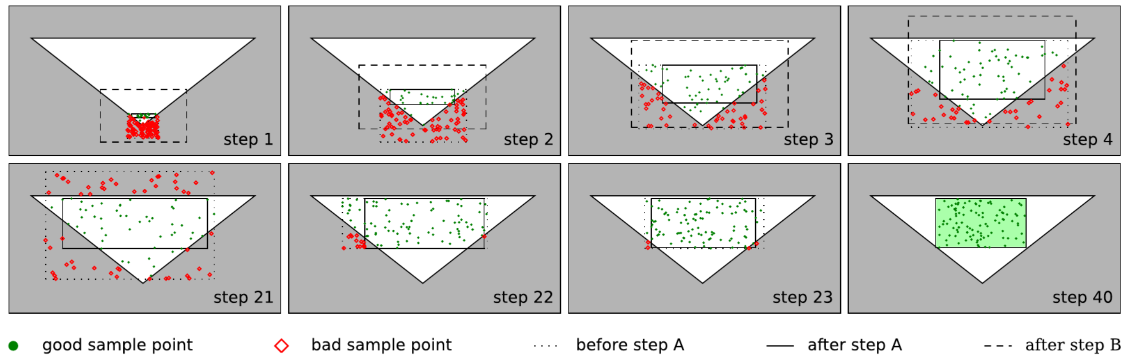


Figure 2.4: Example behaviour of the algorithm proposed by Zimmermann et al. Top row: phase I, bottom row: phase II [33]

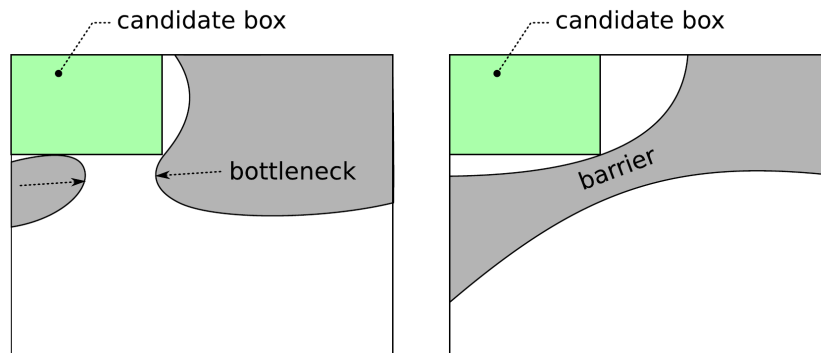


Figure 2.5: Barriers and bottlenecks are a problem for the algorithm by Zimmermann et al. [33].

Even though the proposed solution works with arbitrary and high dimensions and would be applicable, the approach has some drawbacks. The authors state that the algorithm can not guarentee to find the global optimum because of barriers and bottlenecks. The algorithm starts at a good design points and extends its borders from there. It is possible that a box can not grow further if a barrier or bottleneck of bad designs is present. This is visualized in Figure 2.5. The proposed solutions in this thesis should try to circumvent that problem.
Another important problem with the solution of Zimmerman et al. is that it only provides one large solution box. We note that in our case we are also interested in other, possibly much smaller boxes. This provides the engineer with more freedom, as he can choose from multiple design parameter boxes consisting of permissible intervals. Therefore, the solution in this thesis aims to find multiple solution boxes, if possible.

2.3.2 Multiple Solution Boxes

An example case which the solution in our thesis should cover is shown in Figure 2.6. It shows a solution space with three good regions. A naive solution would only find the largest box A. In our solution we aim to find solution boxes of all clusters, i.e. the boxes B and C. Therefore, a processing step to cluster regions of good designs is proposed. In summary, the determination of permissible input intervals consists of two steps:

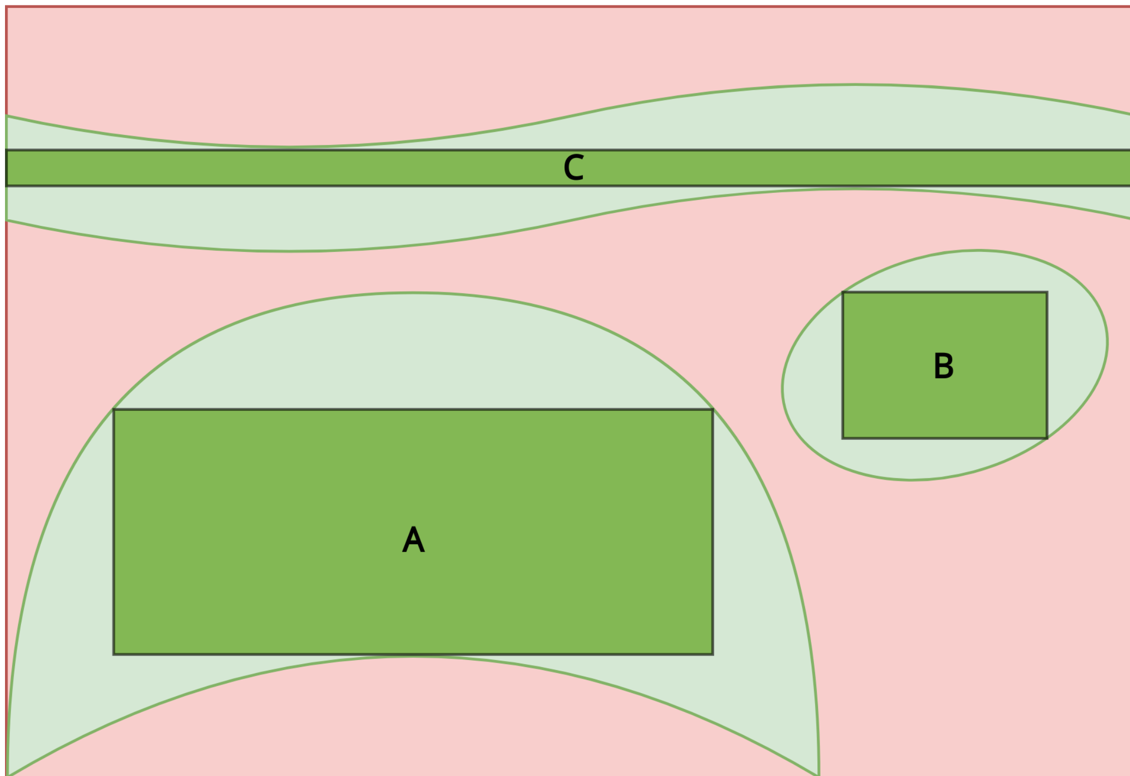


Figure 2.6: Solution space with three good regions.

1. Separating clusters of good designs spatially (Clustering)
2. Determining the largest box with only good designs in those clusters (Box Maximization)

As a side effect we reduce the computation time, because the subdivision of the whole space into clusters results in smaller subspaces, where a solution box can be computed more easily.

2.4 CONCEPT

To summarize, in this chapter, we proposed a general approach to find permissible design parameter intervals on the basis of a dataset of designs. The pipeline of the approach comprises three steps:

1. Outlier Detection
2. Data Augmentation
   (a) Model Generation
   (b) Design Space Sampling
3. Determining Permissible Intervals
   (a) Clustering

(b) Box Maximization

The pipeline is visualized in Figure 2.7.

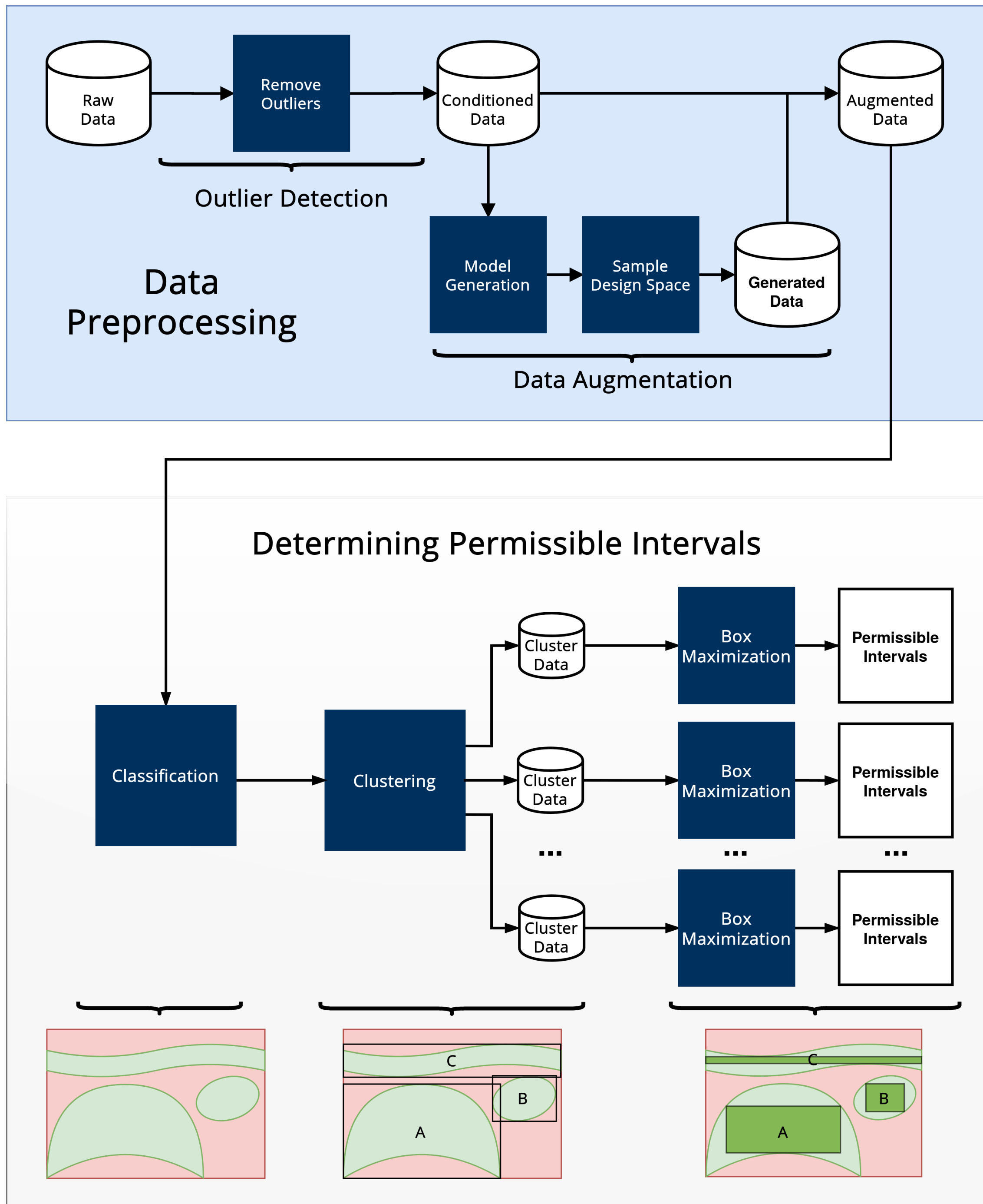


Figure 2.7: Pipeline of the data flow for the approach to determine permissible intervals.

3 METHODS

Based on the pipeline developed in the preceding chapter, the necessary methods to solve the subproblems of the pipeline are revisited. First, in Section 3.1 outlier detection methods are compared and proper methods are selected. Then in Section 3.2 the process of data augmentation is examined, which consists of model generation and sampling methods. Finally, in Section 3.3 we look at methods to determine permissible intervals, which comprises clustering and box maximization.

3.1 OUTLIER DETECTION

Outliers are data points which deviate significantly from other data points in a dataset. Hawkins defines an outlier as '[...] an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism [13]'. Outliers are also referred to as *anomalies*, *discordants*, *deviants* and *abnormalities* [1].
There are two criteria an outlier detection method has to fulfill to fit our purpose: it has to be designed to be usable with multivariate data and can not assume a specific distribution of data. In that way it has to be a generic method for different types of data. Preferrably, it should be interpretable without detailed knowledge about the outlier detection method. This ensures that an engineer has a higher probability to understand how the outlier detection works.
In his comprehensive book *Outlier Analysis* Aggarwal states five basic methods for detecting outliers [1, Section 1.3]: Probabilistic and Statistical Models, Linear Models, Proximity-Based Models, Information -Theoretic Models, High-Dimensional Outlier Detection.
Probabilistic and statistical models assume an underlying distribution of the data. A very common probabilistic method is extreme-value analysis which was originally created for univariate, i.e. one-dimensional data. A disadvantage of the extreme-value analysis is that it is not good at finding outliers in sparse regions in the interior of a dataset. A typical method is the Z-score where the standard deviation from the mean of dataset is computed and used as criteria to classify data points as outliers. We acknowledge that there are ways to use these methods for multivariate data. But since we can neither assume a normal-distribution nor any other distribution, we skip those methods.
The main approach of *linear models* is to project the dataset onto a lower dimensional subspace

which has the least reconstruction error. If the data is projected and reconstructed, outliers have a large reconstruction error because they deviate more profoundly from the pattern of the subspace. Even though the models are linear in principle, it is possible make them nonlinear with PCA and specific kernels. Also, deep learning methods such as autoencoders fall in this category. However, in general the approaches are hard to interpret because the subspace model is a linear combination of possibly positive or negative coefficients, which might not make sense in terms of the context of the data.
Information-theoretic models summarize dataset and encode it. Outliers increase the minimum code length of the summary. A disadvantage in terms of our prototype is the difficulty to interpret the results. Therefore, this technique is not used.
The two remaining methods are *proximity-based* and *high-dimensional* outlier detection. Proximity-based methods are revisited in more depth in the following section. As a representative for high-dimensional outlier detection we explain and implement Isolation Forest in Subsection 3.1.2.

3.1.1 Proximity-Based Methods

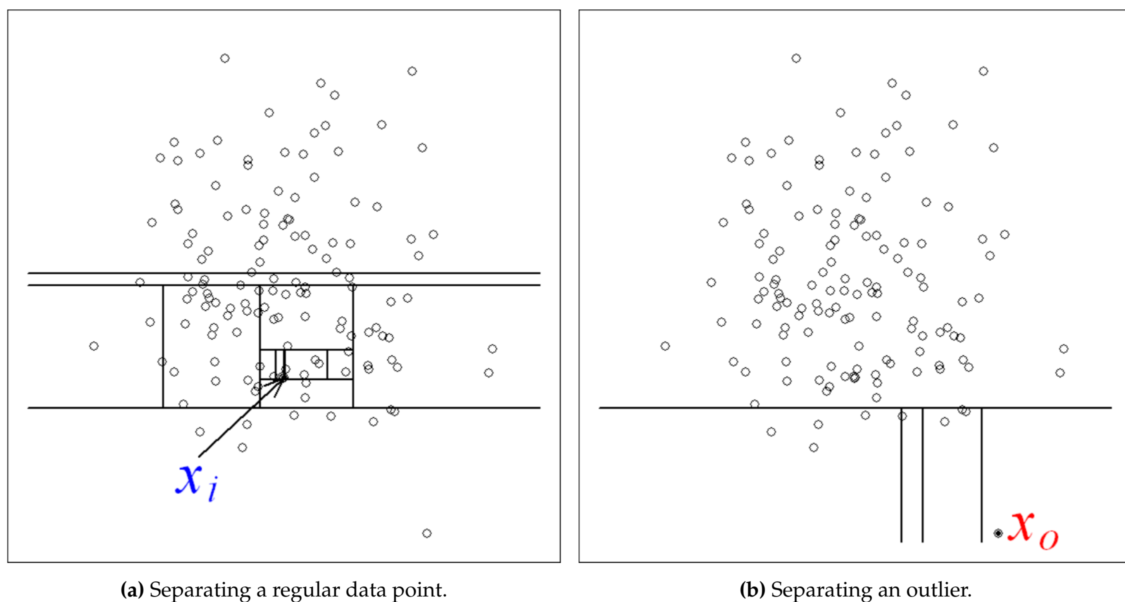
Proximity-based models classify outliers in terms of their locality to other data points. The main advantages are that they are easy to implement and easy to interpret due to the spatial interpretation of the data. Aggarwal distinguishs between three proximity-based methods: distance-based, clustering-based and density-based [1, Chapter 4].

Distance-Based The main approach of distance-based methods is to calculate the distance of data points to their k nearest neighbours. There are three distance scores: *exact*, *average* and the *harmonic* k nearest neighbour score. The exact score adds up the distinct distance values to each k nearest neighbour. The average score averages the exact score by dividing through the number of neighbours. Similarly, the harmonic score calculates the harmonic mean between data points. As a distance measure often the Euclidean distance is used, but this can be defined freely depending on the specific dataset. To distinguish between outliers and regular points, either a *score threshold-based* or *rank threshold-based* method can be used. The score threshold-based method defines a threshold value β. All data points which have distance score of at least β are considered as outliers. In the rank threshold-based method an amount r of outliers is defined. r data points with the highest distance score are considered as outliers.
Distance-based methods are more accurate than the clustring-based and density-based because the have a finer granularity. However, distance-based methods are also less performant, i.e. they have a higher runtime [1, Section 4.1].

Clustering-Based The idea of clustering-based methods is that clustering solves a complementary problem to outlier detection. If a data point does not belong to any cluster, it can be regarded as an outlier. In fact, many clustering algorithms report outliers as a side product. One example is the DBSCAN algorithm which returns a list of noise data points which do not belong to any cluster. DBSCAN is explained in more detail in Subsection 3.3.1. In general, in comparison to distance-based methods, clustering approaches are more performant.

Density-Based In density-based methods outliers are defined in terms of local density in a data space. The main difference between density-based methods and clustering-based methods is that the former partitions the data space and the latter partitions the data. Aggarwal notes that this approach is very similar to the other two methods and that density-based techniques can also be presented as distance-based or clustering-based methods. This is why we concentrate on clustering-based and a distance-based approaches for the software prototype.

3.1.2 Isolation Forest



(a) Separating a regular data point.

(b) Separating an outlier.

Figure 3.1: Isolation forest splitting the data space to isolate data points [19]

Isolation forest is a popular outlier detection algorithm which can handle high dimensions and extremely large datasets [19]. Aggarwal classifies it as an axis-parallel subspace method for high dimensionsal datasets [1, Subsection 5.2.6]. The algorithm has a linear runtime with a low constant and low memory usage.
An isolation forest consists of a constant amount of isolation trees. In a training phase those isolation trees are generated. An isolation tree is generated by splitting the data space in an arbitrary dimension with an arbitrary split value. This is done until all data points are isolated. The key idea is that if the path length of a data point in an isolation tree is lower than the average path length, then the data point has a higher probability to be an outlier. A data point which lies far away from other data points (as in Figure 3.1b) is isolated earlier in the tree (i.e. with fewer splits). A regular point needs many more such splits (as in Figure 3.1a). The average path length of a data point over all isolation trees is used to calculate an anomaly score. The anomaly score is given by

$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}} \tag{3.1}$$

where x is a data point, n the amount of data points in the subsample size, $E(h(x))$ the path length to x averaged over all isolation trees and $c(n)$ the average path length for unsuccessful search in a binary search tree (BST). This score takes on values between zero and one. Liu et al. state that if instances return a score very close to 1, then they are definitely anomalies. If instances have a

score much smaller than 0.5, then they are quite safe to be regarded as normal instances. If all the instances return a score ≈ 0.5, then the entire sample does not really have any distinct anomaly. The algorithm takes two arguments. The first is the amount of trees the forest should generate. Liu et al. show that for most datasets the anomaly scores converges with less than 100 trees. The other argument is the size of the used subsample of the dataset during the training phase. This subset is used to generate the current tree. Subsampling helps avoiding common problems in outlier detection such as swamping and masking. Swamping is the phenomenon of misclassifying regular points as outliers. Masking is the problem that outliers in outlier clusters in large datasets get classified as regular data points. Based on their emprical tests on benchmark datasets, Liu et al. recommend 256 as a subsampling size. To get a binary classification, a threshold value for the anomaly score between zero and one value has to be choosen.

3.2 DATA AUGMENTATION

As explained in Subsection 2.2.2 the data augmentation process consists of two steps: the generation of a model of the dataset and the equidistant sampling of the design space. We close this section with short notes on hyper-parameter optimization and evaluation metrics

3.2.1 Model Generation

Model generation is the first step of data augmentation. A model of the data is used to predict the system response for different combinations of design parameters. This is needed because often sparse areas in the data exist.

Linear Regression

Linear regression models the relationship between independent and dependent variables of a dataset with a linear function. A dataset $D \in R^{n \times d + d_y}$ contains data points which have a feature vector (independent variables) $x_i \in \mathbb{R}^d$ and a output vector (dependent variables) $y_i \in \mathbb{R}^{d_y}$. To simplify the presentation we assume $d_y = 1$ in the following. The generalisation to an arbitrary number of output dimensions is straightforward. A linear function is given by

$$f_c(x) = c_0 + c_1 x[1] + c_2 x[2] + \ldots + c_d x[d] \tag{3.2}$$

The goal is that $y_i^f := f(x_i)$ approximates y_i as good as possible. A requirement is that D contains $n > d + 1$ independent data points. We set up the least-square minimization problem

$$L_D(c) = \sum_{i=1}^{n} (y_i^f - y_i)^2 \rightarrow \min \tag{3.3}$$

Both y_i^f and y_i are known. The only unknown variables are the constants c_j of Equation 3.2. It is not difficult to see that this minimization problem is quadratic with respect to c. Hence,

the problem is reducible to the solution of a system of linear equations. There exist efficient, so called, *direct methods* to solve linear equations based on Cholesky-, QR-, or singular-value-decomposition (SVD) [23, Chapter 10.1]. For that reason, given a dataset, a linear regression model is typically significantly faster to compute than other kinds of ML-models which usually require computationaly more involved algorithms to find the best fit.

Polynomial Regression

Polynomial regression models the same relationship as linear regression except that a polynomial function instead of linear function is used. A polynomial function of degree k can be written as

$$p_c(x) = \sum_{l_1+\ldots+l_d \leq k} c_{l_1 \ldots l_d} x[1]^{l_1} \ldots x[d]^{l_d} \tag{3.4}$$

The problem of polynomial regression is reducible to a linear regression problem. The idea is to extend the feature vectors x_i by the monomials of degree two or higher, appearing in Equation 3.4, to a new *polynomial feature vector*

$$x'_i = (x_i[1], \ldots, x_i[d], \ldots, x[1]^{l_1} \ldots x[d]^{l_d}, \ldots) \tag{3.5}$$

with $x'_i \in \mathbb{R}^{d'}$. Here d' is the number of monomials appearing in definition p_c. We get the linear regression problem by using the polynomial feature vector for linear regression.

$$p_c(x) = f(x') = c'_0 + c'_1 x'[1] + c'_2 x'[2] + \ldots + c'_{d'} x[d'] \tag{3.6}$$

A linear regression problem, again, results in a linear equation system which can be efficiently solved, for example, by using SVD. A simple example of the polynomial features reshaping is shown with $d = 2$ and $k = 2$. The polynomial function and the resulting polynomial feature vector are

$$p_c(x) = c_{00} + c_{10}x[1] + c_{01}x[2] + c_{20}x[1]^2 + c_{11}x[1]x[2] + c_{02}x[2]^2$$
$$x'_i = (x_i[1], x_i[2], x_i[1]^2, x_i[1]x_i[2], x_i[2]^2).$$

Hence, we get:

$$f_c(x') = c'_0 + c_1 x'[1] + c'_2 x'[2] + c'_3 x'[3] + c'_4 x'[4] + c'_5 x'[5]$$

This linear regression problem has five unkown constants (c'_1 to c'_5) for five features. With the help of the well known formula for *combinations with repition* it can be shown that the number of polynomial features is

$$d' = \binom{d+k}{k} - 1 \tag{3.7}$$

Already for small values of k this number is significantly larger than the number d of original features. Thus, in practice the runtime of polynomial regression grows fast with high degrees.

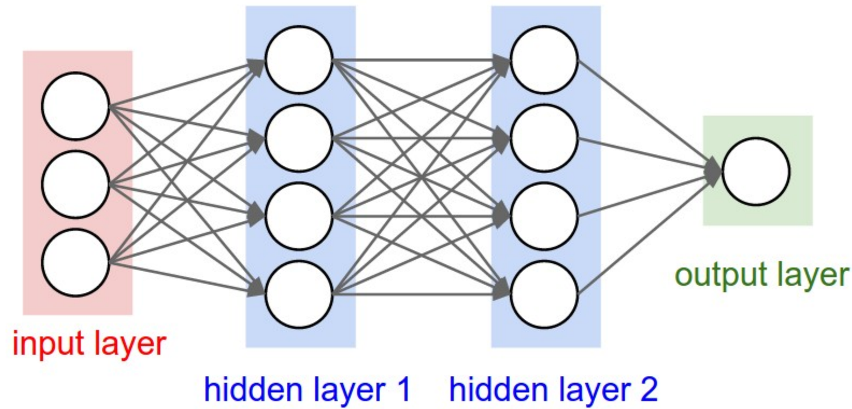


Figure 3.2: A simple neural network with an input layer (three features), two hidden layers and one output layer [15].

Artificial Neural Networks

Artifical neural networks (abbreviated ANN) are inspired by the function of neurons in the brain of vertrebrates [18]. The architecture of a simple ANN is visualized in Figure 3.2. It consists of an input layer, one or several hidden layers and an output layer [22]. The white dots represent the neurons and the arrows between the neurons represent the axons in the brain. The connections of two consecutive layers of an ANN are defined as relatively simple functions. Hence, an ANN is a *composition* of functions which can be expressed as

$$f_c(x) = f_{l+1}(f_l(\dots f_1(x)\dots)) \tag{3.8}$$

with l being the amount of hidden layers. f_{l+1} represents the connections between the last hidden layer and the output layer and f_1 the function between the input layer and the first hidden layer. The functions between the layers are defined as

$$f_r(x) = a(wx + b), a : \mathbb{R} \rightarrow \tag{3.9}$$

where a is a nonlinear activation function, b is a bias term and u_r is the number of units in the layer r. a is a relatively simple function which is applied component-wise on the vectors. Examples are the relu, sigmoid and tanh function. w and b are the new constants which we denoted as c in the case of linear and polynomial regression. Similar to the minimization problem of linear and polynomial regression (Equation 3.3), the goal with artificial neural networks is to minimize a cost function:

$$C(w,d) = \sum_{i=1}^{n}(y_i^f - y_i)^2 \rightarrow \min \tag{3.10}$$

The difference to linear and polynomial regression is that in general this problem does not lead to a quadratic minimization problem with respect to the parameters w and d. A family of iterative methods to solve such general non-linear minimization problems is the gradient descent algorithm and its variations. The so called backpropagation algorithm is an implementation of

gradient descent, designed for neural networks, which leverages their particular compositional structure.
Neural networks have proven very useful for problems like image and speech recognition. However, the training process to generate a model is difficult. The training process requires the engineer to set many hyper-parameters. These hyper-parameters include for example the number of epochs the network should be trained, the learning rate of the gradient descent algorithm, the number of hidden layers and the number of units in those layers. No general rules for the setting of those parameters exist because they depend on the problem at hand. Therefore, we briefly look into techniques for hyper-parameter optimization.

Hyper-Parameter Optimization

The straightforward way to choose appropriate hyper-parameters is to try out different combinations and choose the best one. The exhaustive testing of all hyper-parameter combinations is called *grid search*. For this strategy to work the engineer sets intervals with values for each hyper-parameter. Bergstra et al. state that grid search is the most widely used technique for hyper-parameter optimiziation [3]. It is used since the 1990s [14]. The main problem with grid search is its runtime. The amount of combinations to test grows exponentially with hyper-parameters. In contrast, the *random search* approach does not have this problem. This method fixes the amount of trials to an amount much lower than the trials grid search would use. Bergstra et al. showed empirically and theoretically that random search is more efficient than grid or manual search [3]. Hutter et al. recommend it as a baseline approach [14]. Besides grid and random search other more complex hyper-parameter-optimization strategies exist, e.g. bayesian, gradient-based and evolutionary optimization [14].

Evaluation Metrics

In linear and polynomial regression as well as artificial neural networks the least-square error between the dataset and the model is to be minimized. The lower the least-square error is, the better a model approximates the dataset. Hence, this value can be used to compare and determine the model which fits a given dataset best. However, the least-square error is difficult to use as a general metric. The least-square error depends heavily on the magnitude of values of the dataset in different dimensions which is why the sum of the least-square error can be an arbitrary number. To have a metric which is easier to interpret and can be used to evaluate a model more generally, the R^2 score (coefficient of determination) is used. It is a number for assessing the goodness-of-fit of a regression. It is calculated with the ratio between the sum of squares total (SQT) and sum of squares explained (SQE):

$$R^2 = 1 - \frac{\sum y_i - \hat{y}_i}{\sum y_i - \bar{y}_i} \tag{3.11}$$

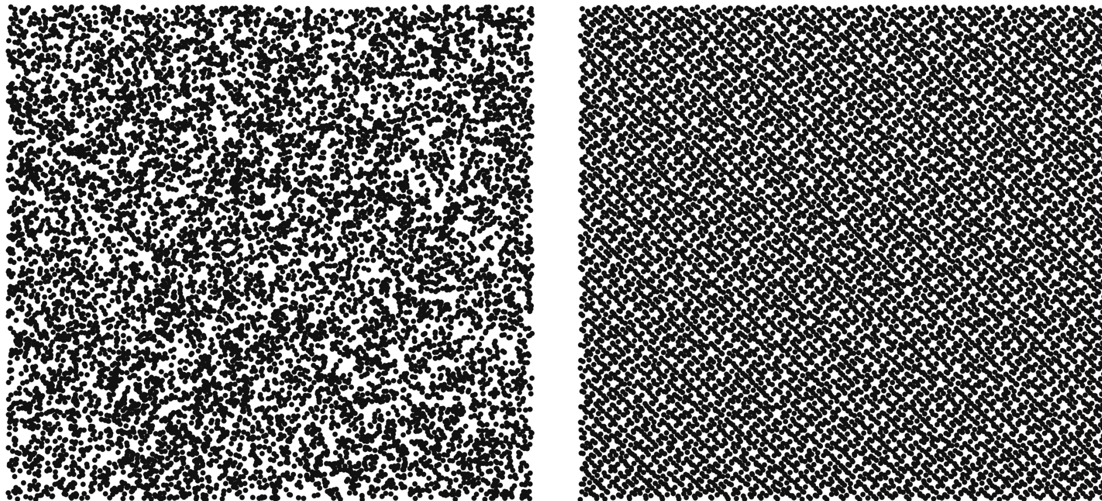
where y_i is the measured value, $\hat{y}_i$ is the value predicted by the model and $\bar{y}_i$ is the average value of the measured values. If the R^2 score is 1, the model fits the dataset perfectly. If it is 0, the model behaves like it would if it always returned the arithmetic mean of for all data points. A negative score signifies that the model performs even worse than arithmetic averaging. Such a model has to be discarded.

3.2.2 Sampling Methods

With a model of the dataset it is possible to generate new data points. Since we want to distribute design points evenly in the whole design space, a method is needed which creates data points in a multidimensional space (hyperrectangle) accordingly. In this thesis, we refer to such a method as a *sampling method*. Likewise, the data points created by a sampling method are called *sampling points*.

Grid Sampling A simple sampling method is to arrange the points in a grid. However, *Grid sampling* has the disadvantage that it constraints the number of points which can be generated for a data space. This is because it has to be possible to take the n-th root of the number of sampling points where n is amount of dimensions of the data space. Otherwise the points can not be distributed evenly.

Random Sampling Another simple sampling method is *Random sampling*. The approach is to distribute points randomly in the data space. Even though this approach allows for the creation of an arbitrary amount of sampling points, the problem is that these points tend to distribute unevenly. This can be seen in Figure 3.3a.



(a) Uniformly distributed pseudorandom data points. **(b)** Data points distributed by Sobol sequence.

Figure 3.3: Comparison of 10 000 data points generated by Random sampling and Sobol sampling.

Sequences of successive points with a preferrably equal distance to their neighbour points are called *low-discrepancy sequences* in mathematics [6]. They are also referred to as *quasi-random number sequences*.

Sobol Sampling An example for such a sequence is the Sobol sequence [28]. Antonov and Saalev proposed an efficient algorithm to generate Sobol sequences by leveraging XOR bit operations [2]. The first 10 000 data points of the Sobol sequence are shown in Figure 3.3b. In comparison to uniformly distributed pseudorandom numbers, the Sobol numbers are visibly more evenly

distributed (Figure 3.3). In this thesis, the Sobol sequence is used to generate sampling points and we refer to this process as *Sobol sampling*.

Besides the Sobol sequence other low-discrepancy sequences exist. A non-exhaustive list includes the Halton sequence [10], the Faure sequence [8] and the van Corput sequence [5].

3.3 DETERMINING PERMISSIBLE INTERVALS

The process of determining permissible intervals consists of a *clustering* step and a *box maximization* step. Clustering groups good designs into regions of good designs, so that an engineer can choose from multiple sets of permissible intervals. Afterwards box maximization algorithms on the clustered data is used to determine permissible intervals for the engineer.

3.3.1 Clustering

Clustering is the process of grouping similar data points together. Aggarwal defines the basic problem of clustering as follows:

> Given a set of data points, partition them into a set of groups which are as similar as possible [1].

In his book *Data Clustering* Aggarwal states that probabilistic / generative models, density-, grid- and distance-based methods for clustering exist.
A popular distance-based algorithm is the k-means algorithm [21]. The only parameter is a number k which defines the number of clusters into which the data points should be grouped. Though widely researched and applied, the algorithm is not applicable in our case because we can not make guess on how many clusters exist in our datasets.
Density-based algorithms comprise algorithms which group data poitns together depending on the spatial distribution. A popular algorithm for density-based clustering is DBSCAN [20].

DBSCAN

DBSCAN takes two arguments: *minPts* and *eps*. *MinPts* is the minimum amount of points which need to be in a distance of *eps* to each other to be regarded as a cluster. This divides the data points into three kinds which can be seen in Figure 3.4. There are core objects (red) which fulfill the aforementioned criteria. Density-reachable points do not fulfill the criteria but are reachable through a core object. They belong the cluster which the core objects span. Finally, there are noise points (blue) which are no core objects and are not density-reachable. The complexity of the algorithm is $O(n \log n)$.

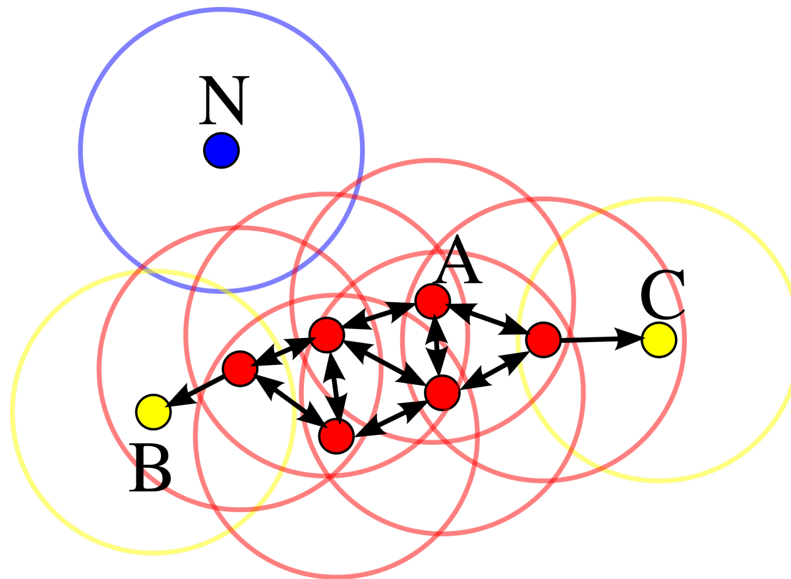


Figure 3.4: DBSCAN divides data points in three categories: core points (red, A), density-reachable points (yellow, B, C) and noise (blue, N) [4].

3.3.2 Box Maximization

The main problem to be solved in this thesis is the box maximization problem developed in Section 2.1. Two approaches will be investigated in this thesis: an exact solution and heuristic solutions.

Exact Solution

In their paper *The Maximum Box Problem and Its Application to Data Analysis* Eckstein et al. propose a branch-and-bound algorithm as an exact solution to a problem which is equivalent to the one proposed in Section 2.1 [7].
The algorithm splits the data space for every negative point along all dimensions and creates subproblems. Figure 3.5 shows an example in two dimensions where there are eight possible ways of constructing subproblem boxes depending on a negative point. Those subproblems are kept in a queue. In each iteration a subproblem box is taken from the queue and checked if it already is a homogenous box, i.e. it only contains positive points (good designs). If so, the subproblem box is compared to the current best solution box and kept if the box is better. If the subproblem box only contains negative points (bad designs), the box is discarded. An upper bound function is implemented, which estimates the best possible solution a subproblem box could create. If the upper bound of a subproblem box cannot achieve a better solution box than the current best box, it is discarded. If not, the subproblem box is further split into subproblems and the newly created subproblems are added to the queue. If the queue is empty, the algorithm terminates.
Eckstein et al. show that the problem of finding the maximum box in a n-dimensional space in general is NP-hard. At the same time they prove that their algorithm will find the largest possible box. Even though the proposed approach suggests intelligent ways of constructing subproblem boxes and choosing negative split points to reduce runtime, in the worst case runtime still grows

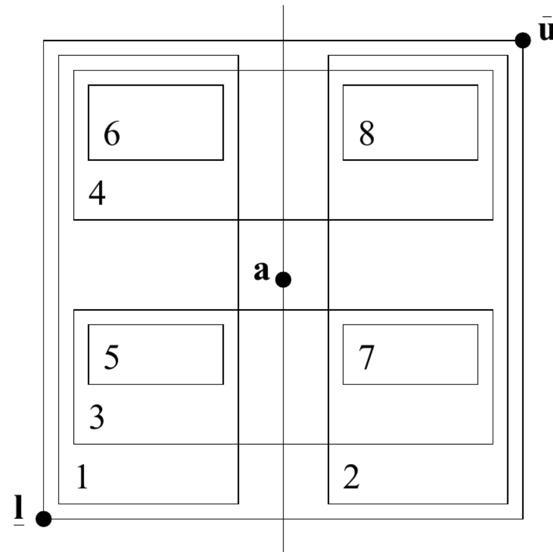


Figure 3.5: A negative point can be used to construct eight different subproblems [7].

rapidly like $O(d^{2n_b})$ with n_b being the amount of negative points (bad designs) and d the dimensionality of the data space. Therefore, it is viable to look for approximate solutions which can calculate solution boxes in arbitrary cases.

Heuristic Solutions

Heuristics can be seen as cost-efficient rule of thumbs which guide actions of an agent [25]. Pearl defines heuristics as

> [...] criteria, methods, or principles for deciding among several alternative courses of action [which] promise to be the most effective in order to achieve some goal. They represent compromises between two requirements: the need to make such criteria simple and, at the same time, the desire to see them discriminate correctly between good and bad choices [25].

The idea of heuristic solutions is to create an algorithm which solves a computational problem where the exact solution is not possible or too difficult to calculate in a reasonable amount of time. In the area of optimization problems, heuristics guiding the process of creating those algorithms are called *metaheuristics*. Sörensen defines:

> A metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms [30].

Many metaheuristics exist; well-known are for example genetic / evolutionary programming, simulated annealing and ant colony optimization [29]. In this thesis we use the metaheuristic simulated annealing to create a box maximization algorithm.

Simulated Annealing is inspired by the annealing process in metallurgy [16]. In that process the temperature of a material is cooled and heated under control to achieve specific properties. The computational heuristic simulated annealing was introduced by Kirkpatrick et al. and first applied to the travelling salesman problem. Simulated annealing works as follows: in every step of the algorithm a neighbour state of the current state is calculated. This neighbour state is used as a new current state with a predefined acceptance probability. The temperature of the system cools down with every step. Neighbour states can vary from the original state in dependance of the temperature. If the temperature is higher, neighbour states are much more different. If it is lower, the neighbour states also differ less. A objective function has to be defined which measures the quality of a state. The last current state is returned.

4 IMPLEMENTATION

In the preceding chapter the methods to solve the main problems of data preprocessing and determining permissible intervals were revisited. In this chapter the implementation of the software prototype is explained in detail. Furthermore, the user workflow is sketched on the basis of screenshots.

4.1 IMPLEMENTATION SPECIFICS

The software prototype is written in the programming language TypeScript and uses the framework Angular 8. The application is a full frontend application. Therefore, no server backend is needed. For the graphical user interface GoldenLayout was used. It allows the arbitrary arrangement of tabs in rows, columns and stacks. The prototype was implemented as a npm package and Angular library which allows for the integration and reuse in multiple applications. The application uses plotly.js to generate graphs and visualizations.
Based on the pipeline (Figure 2.7) developed in Chapter 2, we present the implementation of the distinct steps of the prototype.

4.1.1 Data Preprocessing

Outlier Detection

On the basis of the investigation in Section 3.1 three outlier detection methods were choosen and implemented: Isolation forest, K-Nearest Neighbour and DBSCAN.
Since in many cases the provided data is high dimensional, we implemented isolation forest for outlier detection. The implementation was published in the npm package *isolation-forest* [11]. As stated in Subsection 3.1.2, isolation forest takes two arguments. For the amount of trees we choose 1 000. The subsampling size was set to 256, or the dataset if it is smaller than 256. Since engineers want a binary classification into outliers and regular points, in our implementation there is the

need to set a threshold value on the anomaly score between zero and one. This value has to be set by the engineer and is implemented as a slider panel.
As a distance-based outlier detection method we implemented *k nearest neighbour* approach with average distance scores and a binary classification on the basis of a threshold (see Section 3.1.1). In this implementation data is normalized to the range $[0,1]$ in each parameter dimension. K nearest neighbour needs two parameters; the amount of neighbours k and a threshold β. As a distance function the Euclidean distance is used. Since data is normalized, the largest possible average distance to a neighbour is $\sqrt{d}$ where d is the number of dimensions of the data. This is the maximum value which an engineer can set for this value.
Furthermore, we use the DBSCAN algorithm presented in Subsection 3.3.1 for outlier detection. In our implementation data is normalized to the range $[0,1]$ in each parameter dimension. As stated in Subsection 3.3.1, DBSCAN requires two parameters (eps and minPts) to be set by the engineer. Since data is normalized, the largest distance to a neighbour can be $\sqrt{d}$ with d being the number of dimensions of the data. Therefore, the maximum value an engineer can set for eps is $\sqrt{d}$. An implementation of the DBSCAN algorithm was used from the *density-clustering* npm package [17].

Data Augmentation

For the model generation two different methods were implemented: neural networks and polynomial regression. As a framework for the creation of neural networks Tensorflow.js was used. For the implementation of polynomal regression the npm package *regression-multivariate-polynomial* was used [31]. For the generation of models the data is split into training and test set. For hyperparameter optimization grid search and random search was implemented. To choose the best model a selection process was implemented which chooses the best model depending on the R^2 score on the test set (see Section 3.2.1). For the sampling methods Grid sampling and Sobol sampling was implemented. Grid sampling was implemented without further sources. As a basis for the calculation of the Sobol sequence the npm package *sobol* was used to implement Sobol sampling [24].

4.1.2 Determining Permissible Intervals

Clustering

As a density-based clustering algorithm DBSCAN is used to cluster regions of good designs. DBSCAN needs values for *minPts* and *eps*. Even though the user can change the parameters to his needs, we set a default values for those parameters which worked fine in the tests we conducted. We set $minPts = 2$ and $eps = 2\sqrt[d]{\#designs}$ where d is the number of dimensions and # designs the amount of design points. The factor two serves as confidence factor to rather include than to exclude neighbour designs.

Box Maximization Algorithms

Exact Max Box In Subsection 3.3.2, we presented an exact solution to determine permissible intervals. The algorithm was implemented as stated in the paper by Eckstein et al. [7]. From now on we refer to this algorithm as *Exact Max Box*. Even though the box maximization problem is NP-hard, we implemented it to have an upper bound for the best possible solution, at least for small instances of the problem.

Since an exact solution for a 'large' instance of an NP-hard problem is practically uncomputable, we implemented two heuristic algorithms which merely try to find an approximate solution. The simulated annealing metaheuristic is used construct an algorithm which can provide solution boxes in more difficult cases, e.g. in high dimensions and with many design points. We refer to this algorithm as *Anneal Max Box*. Secondly, a naive random algorithm *Random Max Box* was implemented which serves as lower bound benchmark for the Anneal Max Box algorithm.

Anneal Max Box The simulated annealing metaheuristic described in Section 3.3.2 was used to create a heuristic algorithm. The algorithm needs the amount of iterations as an argument. The amount of iterations is used for the calculation of the current temperature in each iteration. As a start temperature t_0 the amount of desgin points is used. The temperature decreases exponentially with each iteration i:

$$t(i) = t_0 \, e^{-10 \frac{i}{i_{max}}}$$

where i refers to the current iteration and i_{max} is the amount of iterations the algorithm performs. In the context of our box maximization problem a state is a box. A neighbour state is a box which is derived from the current box. In addition to the current box we save the best box ever discovered. We derive neighbour boxes both either of the current box or the best box with equal probability. A new neighbour box is generated from an existing box by translation and scaling. The maximum scaling factor is adjustable through a parameter. By default the box is scaled in the interval $[0.66, 1.5]$. The translation depends on the current temperature and is drawn from a normal distribution with standard deviation of $1.5 \, \frac{t(i)}{t_0}$ which is multiplied by the length of the data space in the translation dimension. The acceptance probability for changing the current box to a new box is given by

$$P(B, B', t) = \begin{cases} 1 & \text{, if } n' < n \\ 1 & \text{, if } n' = n, p' \geq p \\ e^{-\frac{n'-n}{k_1 t}} \, e^{-\frac{p-p'}{k_2 t}} & \text{else} \end{cases}$$

where B is the current box, B' is the new derived neighbour box of B, n and n' are the number of negative designs in B and B' respectively and p and p' the amount of positive desings in those boxes. The acceptance probability can be explained as follows: if the new box ist better than the current box, we accept it. Even if the box is worse, we sometimes accept it. The probability to accept a worse box approaches zero if the temperature approches zero. The particular form of the else-case in the definition of the acceptance probability is motivated by the formula for the Boltzmann distribution from statistical mechanics. The values k_1 and k_2 were set to one in this implementation.

After i_{max} iterations the algorithm terminates and returns the best box.

Random Max Box chooses for each dimension two values randomly. The lower value is assigned as a minimum value for that dimension and the higher value as a maximum value. One parameter is the number of iterations. For every iteration a random box is generated and compared to the current best solution box. If the newly created box has a higher box fitness, it replaces the current best solution box.

Both heuristics have an in-built mechanism to run the algorithm repeatedly. This additional parameter is called *trials*. By default, the amount of trials is set to $3d$ where d is the amount of design parameter dimensions. This helps the algorithms to converge.
Furthermore, the two heuristics do not have the problem Zimmermann et al. stated in their solution: in both algorithms the boxes are able to jump to arbitrary positions in the design space. Thus, the barrier / bottleneck problem described in Subsection 2.3.1 does not occur with these algorithms.
All three algorithms are written conservatively. This means that every solution box needs supporting design points. A box with a specific minimum or maximum value has to posess at least one value of one design point with that value in each dimension. A box is not allowed to be bigger than its points. The algorithms are implemented to fulfill this condition.
All algorithms use the box fitness from Equation 2.4 as a metric to determine the best solution box by comparing boxes based on their box fitness.

4.2 USER WORKFLOW

In this section we describe how the software prototype can be used. Figure 4.1 shows the graphical user interface of the software protoype. All main actions can be triggered by right-clicking on the table on the left to open a context menu. The context menu contains entries for opening pop-up windows (outlier detection, plot creation) and creating new tabs (model generation, determining permissible intervals). In the beginning the table is empty and the user loads a CSV files into the application via the entry *Import Data*. The application then displays the dataset in the table and generates a scatterplot matrix as an overview which can be seen in Figure 4.1 on the bottom in the middle. In general, the user can resize and arrange tabs to suit his needs. The user can create custom plots like the 3D plot on the top in the middle. To do this he chooses the entry *Create Plot* in the context menu. All plots are connected to the table via a linking and brushing mechanism. Selections in the table, highlighted with a teal background color, are highlighted in orange in the plots. Furthermore, outliers are highlighted in the table and in the plots in red color. Outliers can be either detected with the outlier detection methods via *Detect Outliers* or set manually through the checkbox in the data table. All outliers can be conveniently deactivated. If data rows are deactivated, they are removed from all plots and not used for any calculations of the software. It is also possible to deactivate data rows manually.
The model generation component can be seen on the right in a dedicated tab. On top it shows all available models and detailed information about them. An important measure is the R^2 score which is shown in either red, orange, yellow or green depending on the quality of the model. Displayed below is an assessment of the current active model with a predicted / measured graph as a further indicator of model quality.

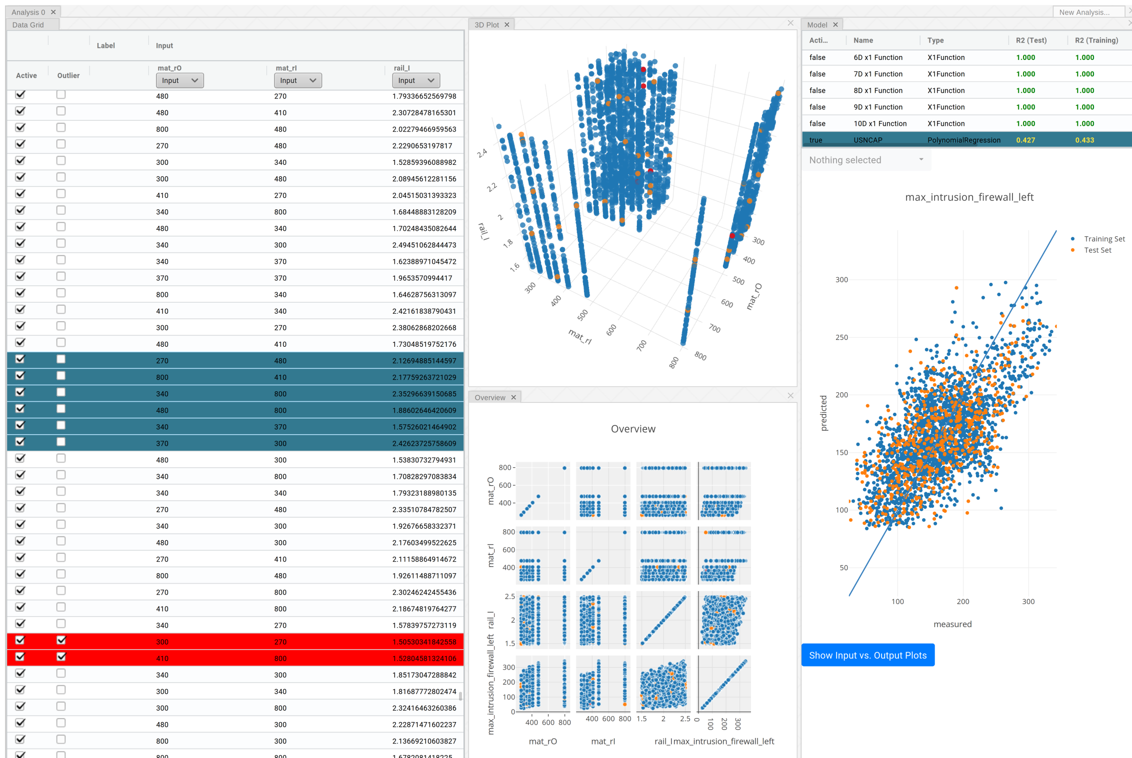


Figure 4.1: Typical GUI view of the software prototype.

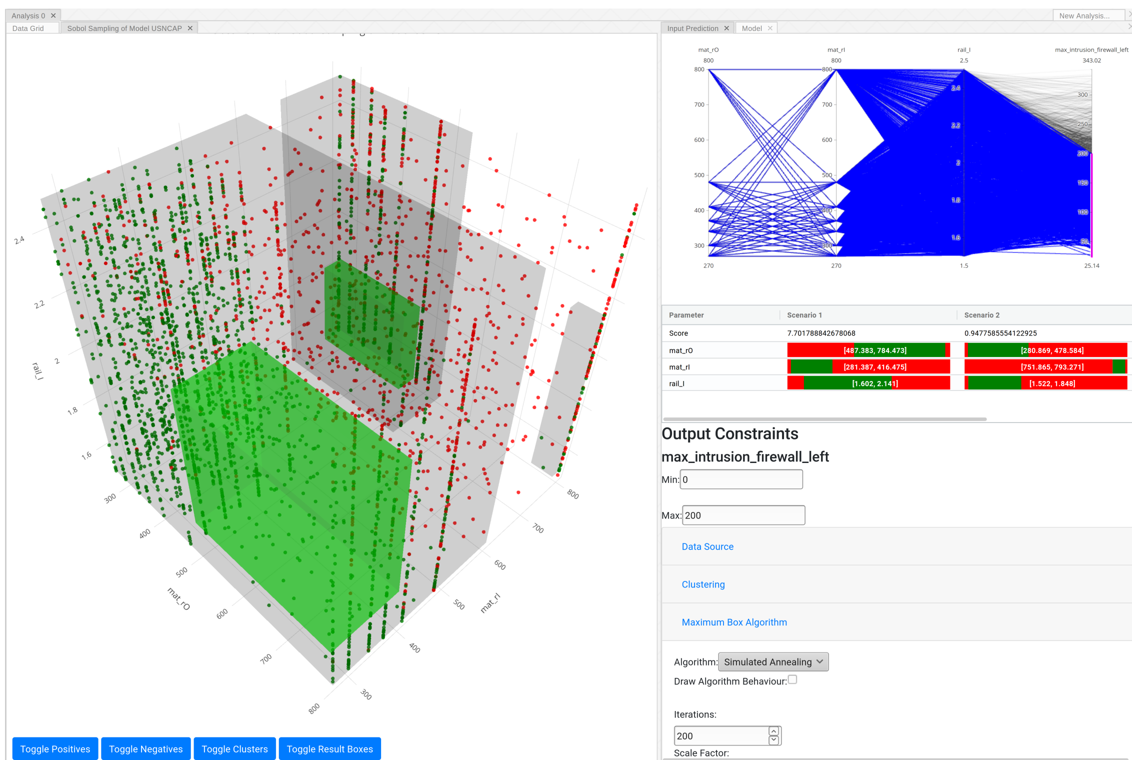


Figure 4.2: Left: Visualisation of the design space with good and bad designs.
Right: GUI for determining permissible intervals.

Figure 4.2 shows a different view of the application which focuses on the process of determining permissible intervals. On the right the component for determining permissible intervals is displayed. On the top a parallel coordinates plot shows the real design points which generated the current active model. On the system responses axes the user can set constraints. Constraints can also be set on the input elements below the table. The table shows the calculated solution boxes and visualizes each design parameter dimension with red and green bars. Additionally, the permissible intervals are displayed with numbers on top of the bars. Each scenario also shows the box fitness of the solution box. On the bottom an accordion menu contains the specific setting options for data augmentation, clustering and the box maximization algorithms. If desired, it is possible to visualize the behaviour of the box maximization algorithms in a dedicated tab. On the left the solution space with good and bad designs is displayed. This works for one, two and three dimensions. For solution spaces with higher dimension, the first three design paramter dimensions are displayed. The graph allows to toggle positive and negative designs as well as cluster boxes and solution boxes. In complex solution spaces with three or more dimensions this helps to interpret the results.

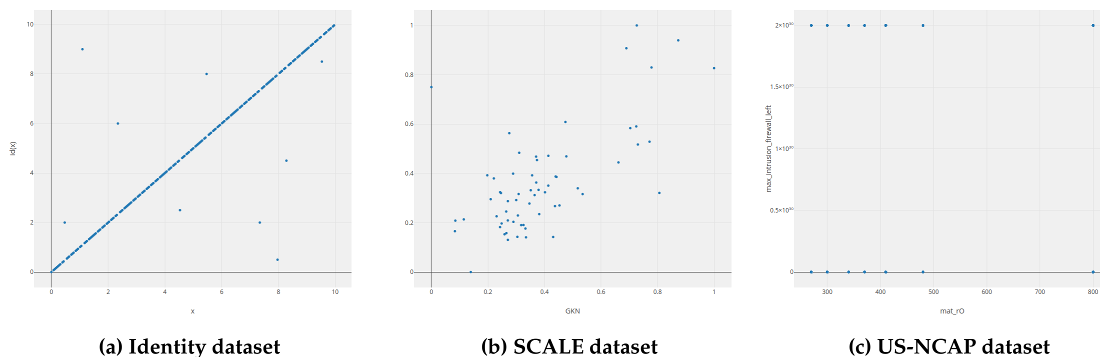
5 COMPONENT-WISE EVALUATION

In the preceding chapter we described the implementation of the software prototype. In this chapter outlier detection, clustering and box maximization methods are evaluated. An overview over the used datasets and functions for data generation can be found in Appendix A.

5.1 OUTLIER DETECTION

Outlier detection is the first data preprocessing step of the prototype (see Figure 2.2). To evaluate the outlier detection methods, three datasets are being used: the Identity dataset (A.2.2), the SCALE dataset (A.1.1) and the US-NCAP dataset (A.1.2). All datasets contain outliers:

1. The **Identity dataset** is shown in Figure 5.1a. It has one design parameter and one system response. Nine outliers deviate from the diagonal line.
2. The **SCALE dataset** has twelve design parameters and two system responses. One outlier is known. In the plot in Figure 5.1b it has the value $(0, 0.75)$.
3. The **US-NCAP dataset** consists of 27 design parameters and twelve system responses. It has 118 known outliers which are visualized in the top row in Figure 5.1c. The outliers have



(a) Identity dataset **(b) SCALE dataset** **(c) US-NCAP dataset**

Figure 5.1: Three evaluation datasets and their outliers.

a value of 2×10^{30} on their system responses parameters. They divide the design space heavily in two extremes.

Since all outlier detection methods need parameters to yield results, in this evaluation good values are choosen in a fixed amount of time with trial and error. The quality of the results is measured in *Precision*, *Recall* and *F1-Score*:

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{5.1}$$

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{5.2}$$

$$F1Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{5.3}$$

True Positives are outliers which get classified as outliers by the detection algorithm. *True Negatives* are regular points which are correctly classified as regular points. *False Positives* are regular points which get mistakenly classified as outliers. *False Negatives* are outliers which do not get classified as outliers. In our context, *precision* is the fraction of correctly classified outliers among all design which were classified as outliers. *Recall* is the fraction of correctly classified outliers from the amount of correctly classified outliers and the regular points which were mistakenly classified as outliers. The *F1 Score* is the harmonic mean between precision and recall.

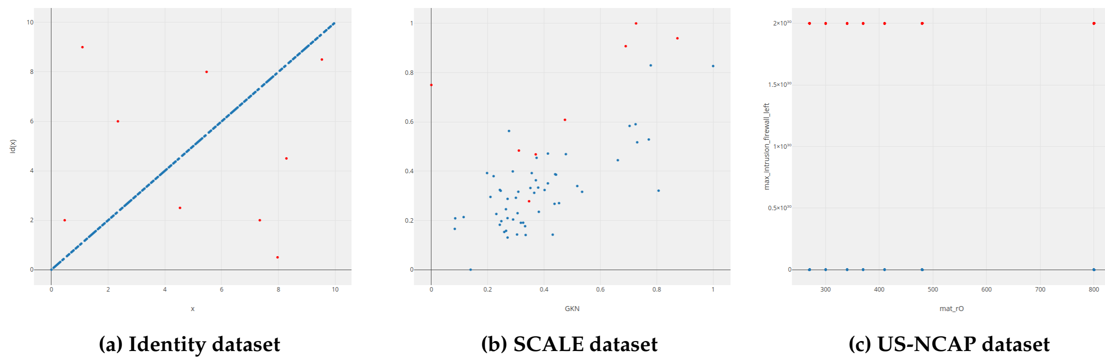
5.1.1 K Nearest Neighbour

In the **Identity dataset** k nearest neighbour outlier detection finds all outliers with k = 4 and distance threshold = 0.05. Thus, precision, recall and F1 score are one. In the **SCALE dataset** it finds all outliers with parameters k = 2 and distance threshold = 0.68. It misclassifies 7 regular points as outliers. The resulting precision is 0.125, recall is 1 and F1 score is 0.18. In the **US-NCAP dataset** it finds all outliers with parameters k = 200 and distance threshold = 2.75. Thus, precision, recall and F1 score are one.
Results are visualized in Figure 5.2 and summarized in the table below:

	kNN		
Dataset	Precision	Recall	F1 Score
Identity	1	1	1
Scale	0.125	1	0.18
US-NCAP	1	1	1

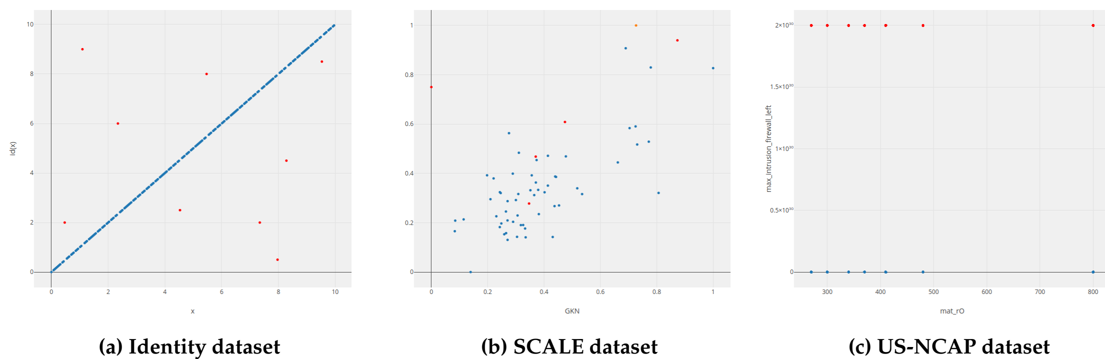
5.1.2 DBSCAN

In the **Identity dataset** DBSCAN clustering outlier detection finds all outliers with minPts = 4 and eps = 0.05. Thus, precision, recall and F1 score are one. In the **SCALE dataset** it finds all outliers with minPts = 2 and eps = 0.68. Five regular points get misclassified as outliers (False Positives). The resulting precision is approximately 0.17, recall is one and F1 score is approximately 0.29. In



(a) Identity dataset (b) SCALE dataset (c) US-NCAP dataset

Figure 5.2: Results of **K Nearest Neighbour** outlier detection on three datasets. Parameters were choosen so that all outliers were found (i.e. Recall = 1).



(a) Identity dataset (b) SCALE dataset (c) US-NCAP dataset

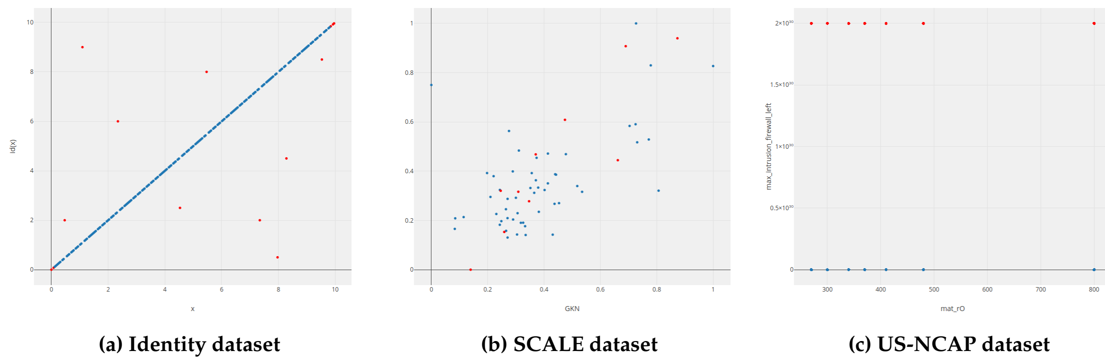
Figure 5.3: Results of **DBSCAN clustering** outlier detection on three datasets. Parameters were choosen so that all outliers were found (i.e. Recall = 1).

the **US-NCAP dataset** all outliers are found with minPts = 120 and eps = 3. Hence, precision, recall and F1 score is one.
Results are visualized in Figure 5.3 and summarized in the table below:

	DBSCAN		
Dataset	Precision	Recall	F1 Score
Identity	1	1	1
Scale	0.17	1	0.29
US-NCAP	1	1	1

5.1.3 Isolation Forest

In the **Identity dataset** isolation forest finds all nine outliers given an intensity value 0.56. It misclassifies five regular points as outliers at the ends of the distribution (False Positives). Therefore, precision is 0.64, recall is one and F1 score is 0.78. We see that isolation forest performs bad at the beginning and the end of data spaces. These are the areas where the misclassification happens. In the **SCALE dataset** isolation forest finds the outlier but misclassifies 7 regular points as outliers. The resulting precision is 0.125, recall is one and F1 score is 0.18. In the **US-NCAP dataset** isolation forest finds all 118 outliers with an intensity value 0.44. Hence, precission, recall and F1 score are one.
Results are visualized in Figure 5.4 and summarized in the table below:



(a) Identity dataset (b) SCALE dataset (c) US-NCAP dataset

Figure 5.4: Results of **Isolation Forest** outlier detection on three datasets. Parameters were choosen so that all outliers were found (i.e. Recall = 1).

	Isolation Forest		
Dataset	Precision	Recall	F1 Score
Identity	0.64	1	0.78
Scale	0.125	1	0.18
US-NCAP	1	1	1

5.1.4 Summary

	kNN			**DBSCAN**			**Isolation Forest**		
Dataset	Prec.	Rec.	F1	Prec.	Rec.	F1	Prec.	Rec.	F1
Identity	1	1	1	1	1	1	0.64	1	0.78
Scale	0.125	1	0.18	0.17	1	0.29	0.125	1	0.18
US-NCAP	1	1	1	1	1	1	1	1	1

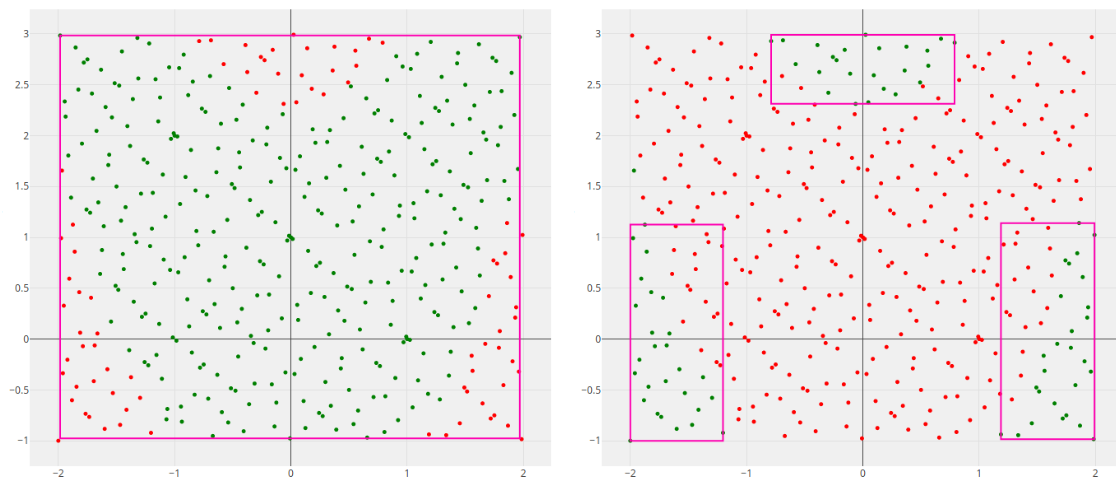
The evaluation of outlier detection methods yields the following insights. First, parameters heavily depend on the dataset and the outliers to be found. Thus, there is no general rule for parameter selection. Also, the higher the amount of dimensions of the dataset in which outliers should be found, the higher the values of distance for kNN and eps for DBSCAN have to be choosen. This is consistent with the observation in Section 4.1.1 that the maximum distance value for kNN and the eps value for DBSCAN grows with $\sqrt{d}$, where d is the number of dimensions of the data.
More dimensions with noise make it more difficult to differentiate between outliers and regular points. All outlier detection methods had problems to classify the outlier of the SCALE dataset (Subsection A.1.1). This fits because the dataset contains real crash test data which can contain noise in the data. In general, we see that all outlier detection methods perform equally good on the Identity and US-NCAP dataset except for isolation forest on the Identity dataset. This fits to the fact that isolation forest is a high-dimensional outlier detection method, which can not handle two-dimensional data well.
For an engineer, it is difficult to choose parameters for kNN and DBSCAN because in both cases the two parameters are interacting with one another and knowledge about the underlying algorithm is necessary. Isolation Forest on the other hand only needs an intensity value which can be understood and used intuitively with a slider panel. For practical reasons, therefore, Isolation Forest is favoured.

5.2 CLUSTERING

Clustering is the second step in determining permissible intervals (see Figure 2.7). It enables the possibility to provide the engineer with more than one set of permissible intervals, i.e. more than one solution box. As described in Subsection 3.3.1, the prototype uses DBSCAN to find clusters of good regions.
The Rosenbrock function was used to generate 400 data points with Sobol sampling. The function is defined as:

$$f(x,y) = (a-x)^2 + b(y-x^2)^2 \tag{5.4}$$

with $a = 1$ and $b = 100$. Parameters x and y are design parameters and the function value is the system response. We use two constraints on the system response: $0 \leq f(x,y) \leq 500$ and $500 \leq f(x,y) \leq 4000$. As a result, we get the distribution of good and bad design points seen in Figure 5.5. In the first case one cluster should be found, while in the second case there are three clusters to be found. DBSCAN, with the heuristically choosen values for eps and minPts (as defined in Section 4.1.2), finds the clusters successfully. In Figure 5.5, bounding boxes are drawn around the clusters. A *bounding box* of a set of data points is the box of points which is spanned by the minimum and maximum values of all data points in each dimension.



(a) Design space with one good region. Constraints: $0 \leq f(x,y) \leq 500$

(b) Design space with three good regions. Constraints: $500 \leq f(x,y) \leq 4000$

Figure 5.5: DBSCAN finds all clusters properly. Bounding boxes (pink) are drawn around clusters of good designs.

Furthermore, in our tests it came to know that DBSCAN clustering took more than 2 hours to terminate with 50 000 data points. Since DBSCAN is widely used and a well-researched algorithm, we refer to Subsection 3.3.1 and references therin for further information about its performance, runtime, advantages and disadvantages.

5.3 BOX MAXIMIZATION

Box maximization is the third step in determining permissible intervals (see Figure 2.7). In this section we test the three proposed box maximization algorithms: Exact Max Box, Anneal Max

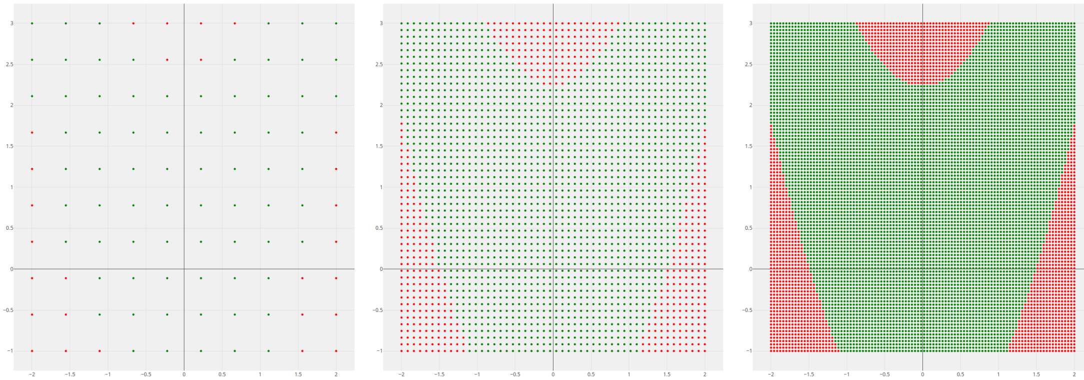


Figure 5.6: Three design spaces sampled using the Rosenbrock function with 100, 500 and 10 000 data points respectively and constraints $0 \leq f(x,y) \leq 500$. The largest possible solution box can be approximated more accurately if the underlying function is approximated by more designs.

Box and Random Max Box.
In this evaluation box maximization problems are generated using functions. First, a function is used to create data points. Then appropriate constraints on the system response are choosen depending on the focus of the experiment. Because the underlying function is known, this approach allows the analytical computation of the largest solution box which is used as a benchmark.
We also vary the amount of data points to find out how many data points per dimension are necessary to create solution boxes with a high box fitness. Clearly, solution boxes heavily depend on the amount of data points. In a design spaces with more designs, solution boxes make us of more information. The border between good and bad designs gets represented more accurately. Figure 5.6 shows this using the Rosenbrock function.
Likewise we test two sampling methods for distribution of data points in the design space: Grid sampling and Sobol sampling. With it we see how solution boxes are affected by the way points are distributed in the design space.

Therefore, in this evaluation the following four independent variables are varied:

- Sampling Points (i.e. number of designs)
- Sampling Methods (Grid sampling, Sobol sampling)
- Box Maximization Problems (Linear Problems, Nonlinear Problems)
- Box Maximization Algorithms (Exact Max Box, Anneal Max Box, Random Max Box)

The first dependent variable which is measured in this evaluation is the box fitness. The second dependent variable is the runtime it takes for the algorithm to compute a solution box. Hence, there are two dependent variables:

- Runtime (in seconds)
- Box Fitness (Range: [0,100])

Tests were carried out on a Intel(R) Core(TM) i5-4210M CPU @ 2.60GHz, 16GB RAM.

5.3.1 Linear Problems

A linear box maximization problem is created by using a sum function. It is defined as

$$f(\mathbf{x}) = \sum_{i=1}^{n} x_i, x_i \in [0,1], n \in \mathbb{N}_{\neq 0} \tag{5.5}$$

with n being the number of dimensions of the vector **x** (Section A.2.1).

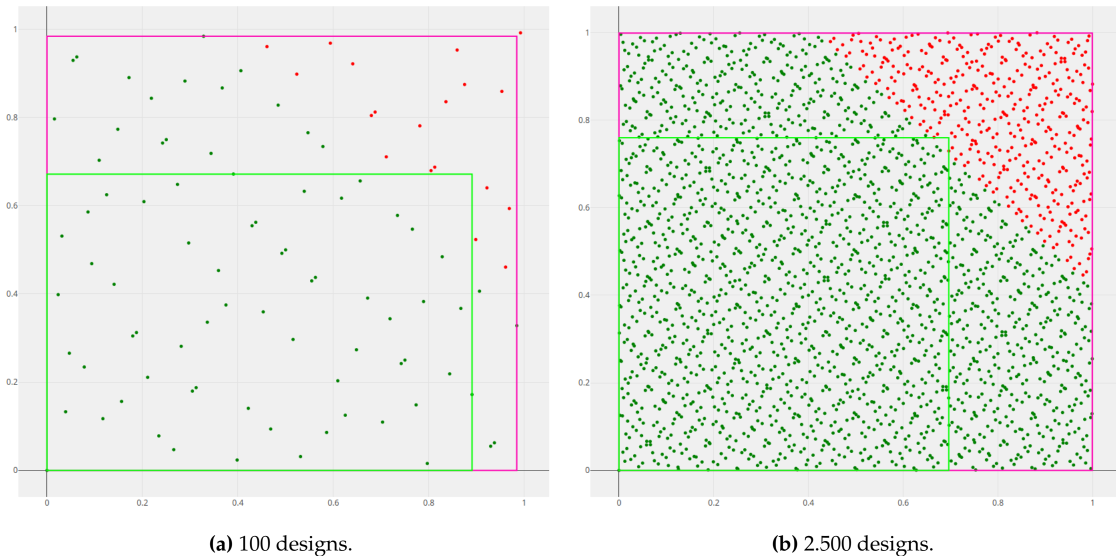
A constant value for the box fitness should be specified which is then used as an analytical benchmark. Therefore, we have to set a constraint on the sum function value (system response) in a way that the box fitness equals our desired value. This constraint differs with the amount of dimensions. In Section A.3 we assume $f_c^l = 0$ and show that the upper constraint f_c^u depends on the box fitness μ with $f_c^u = p\sqrt[p]{\frac{\mu}{100}}$. Thus, the constraints on the sum function in general needs to be

$$f_c = \left[0, p\sqrt[p]{\frac{\mu}{100}}\right] \tag{5.6}$$

where μ can be defined as one wishes.

2D Sum Function

In this section we use the two-dimensional sum function (i.e. p = 2) and we want to specify a box fitness of 50. With Equation 5.6 the necessary constraint is $2\sqrt{\frac{50}{100}} \approx 1.414213562373095$. In Figure 5.7 the solution space is visualized. Runtime constraint is set to one hour. The heuristic algorithms are executed 9 times in a row to calculate an average box fitness value.



(a) 100 designs. **(b)** 2.500 designs.

Figure 5.7: Data: 2D Sum Function, **Algorithm**: Exact Max Box, **Constraints**: $0 \leq f(x) \leq 1.4142$

Exact Max Box Results are shown in Figure 5.8. In general, Sobol sampling seems to lead to an slight overestimation and Grid sampling to a slight underestimation of the solution box. In

Figure 5.7a the best solution box visibly enters the triangular area on the top right. With more sampling points this behaviour can be minimized (Figure 5.7b). Apparently, Exact Max Box takes much more time to terminate for a high amount of data points, for example, approximately 17 minutes with Sobol sampling and 4.900 data points or 21 minutes with Grid sampling and 40 000 data points). Here, Grid sampling terminates faster than Sobol sampling. The algorithm did not terminate within an hour for 10 000 and 40 000 data points with Sobol sampling, hence this data is missing in the diagram.

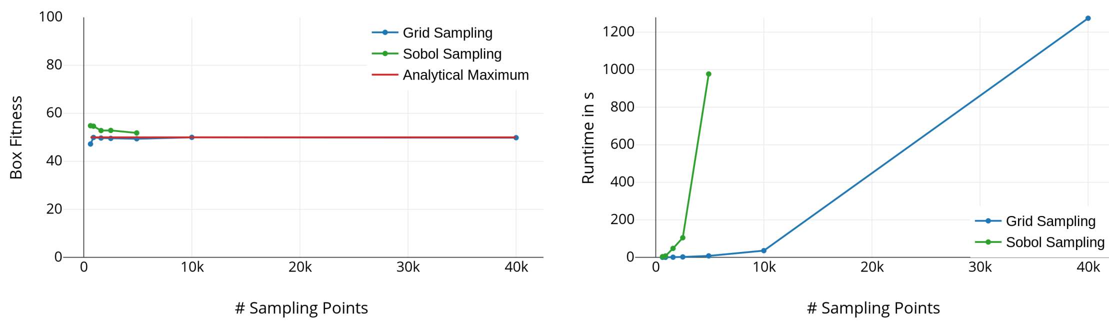


Figure 5.8: Data: 2D Sum Function, **Algorithm**: Exact Max Box, **Constraints**: $0 \leq f(x) \leq 1.4142$

Anneal Max Box Results are shown in Figure 5.9. In general, the computed solution boxes are very good. With 625 designs the boxes are almost as large as the analytical maximum. As with Exact Max Box, Sobol sampling leads to a slight overestimation and Grid sampling to a slight underestimation of the solution box. There is no difference between the runtime for the two sampling methods. Both grow linearly with the amount of sampling points and stay in the range from 5 to 28 seconds.

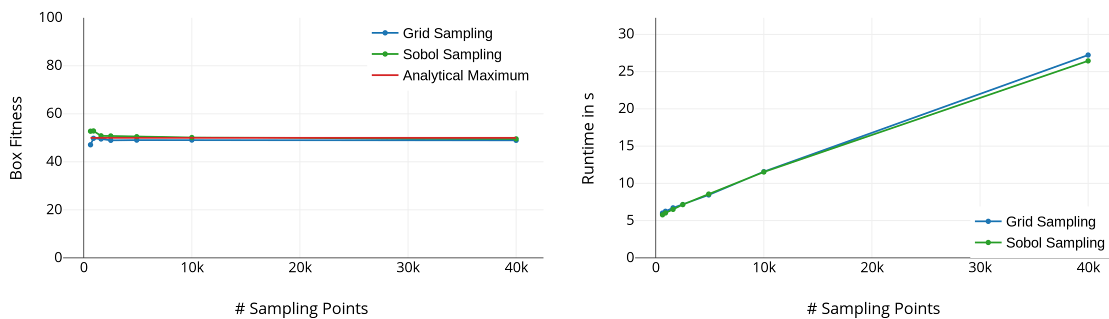


Figure 5.9: Data: 2D Sum Function, **Algorithm**: Anneal Max Box, **Constraints**: $0 \leq f(x) \leq 1.4142$

Random Max Box Results are shown in Figure 5.10. The results are moderate. Box fitness converges around 40 and differs in each trial. As with the Anneal Max Box, there is no difference between the runtime of the two sampling methods. Both grow linearly and stay in the range from 5 to 29 seconds.

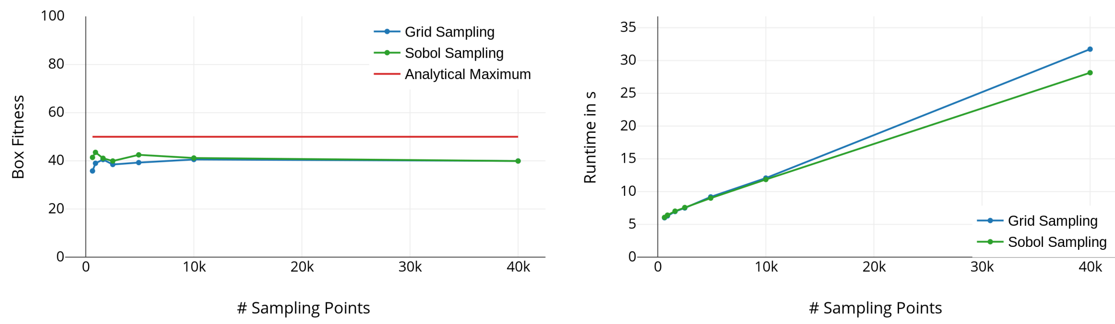
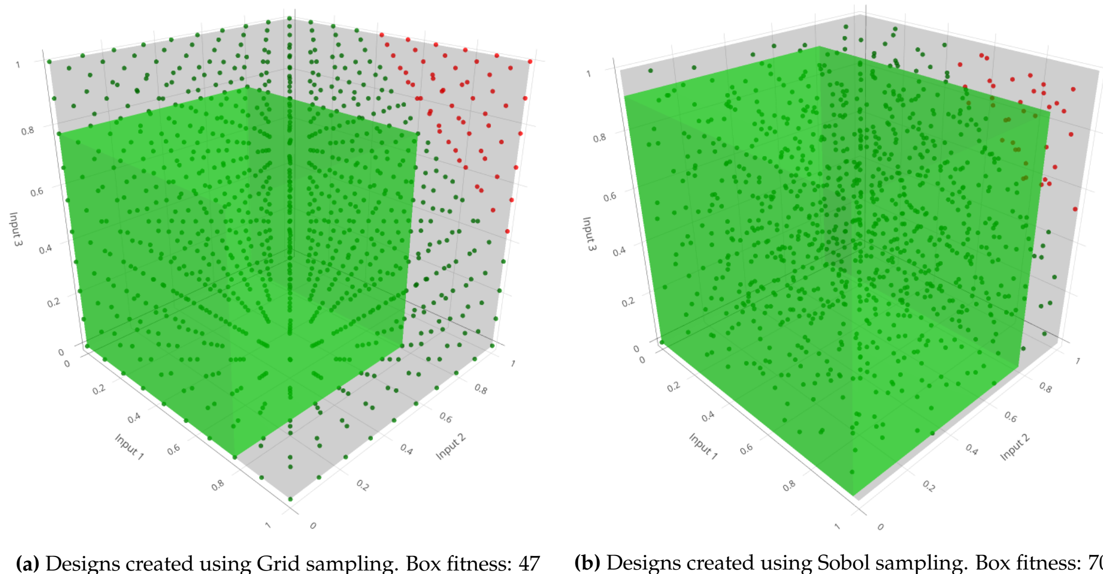


Figure 5.10: Data: 2D Sum Function, **Algorithm**: Random Max Box, **Constraints**: $0 \leq f(x) \leq 1.4142$

3D Sum Function

In this section we use the three-dimensional sum function (i.e. p = 3). With Equation 5.6 the necessary constraint is $3\sqrt[3]{\frac{50}{100}} \approx 2.3811015779522991$. In Figure 5.11 the solution space is visualized. Runtime constraint is set to one hour. The heuristic algorithms are executed 9 times in a row to calculate an average box fitness value.



(a) Designs created using Grid sampling. Box fitness: 47 **(b)** Designs created using Sobol sampling. Box fitness: 70.9

Figure 5.11: Data: 3D Sum Function, **Algorithm**: Exact Max Box, **Constraints**: $0 \leq f(x) \leq 2.3811$, **Sampling Points**: 1 000

Exact Max Box Most notably, the runtime of the problems sampled with Sobol sampling increase with more sampling points more strongly than with Grid sampling. Like in two dimensions, the Exact Max Box overestimates boxes if Sobol sampling is used. The difference between Grid and Sobol sampling can be seen in Figure 5.11 with 1 000 sampling points. The largest Sobol box protrudes more significantly into the tetrahedron in the upper right corner than the largest Grid box. Note, that even though this is the case, the largest box still does not contain any bad design. The usage of Grid sampling is more conservative and results in boxes which are very close to the analytical maximum.

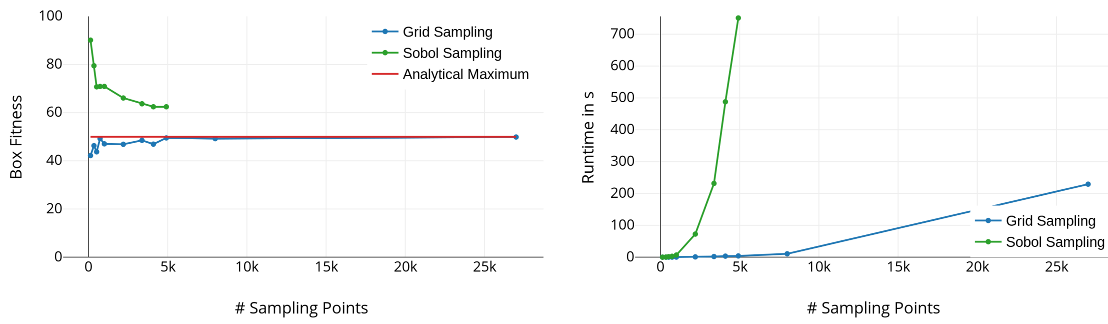


Figure 5.12: Data: 3D Sum Function, **Algorithm**: Exact Max Box, **Constraints**: $0 \leq f(x) \leq 2.3811$

Anneal Max Box gives good results. Sobol sampling approximates the analytical maximum from above and Grid sampling from below. For Sobol sampling it starts with a box fitness of 88 and for Grid Sampling with 36. Both converge to the analytical maximum of 50. With 25 000 design points, Sobol sampling leads to a box fitness of 52 and Grid sampling has a box fitness of 48. The runtime grows linear with the amount of sampling points (range: 8 - 32 seconds).

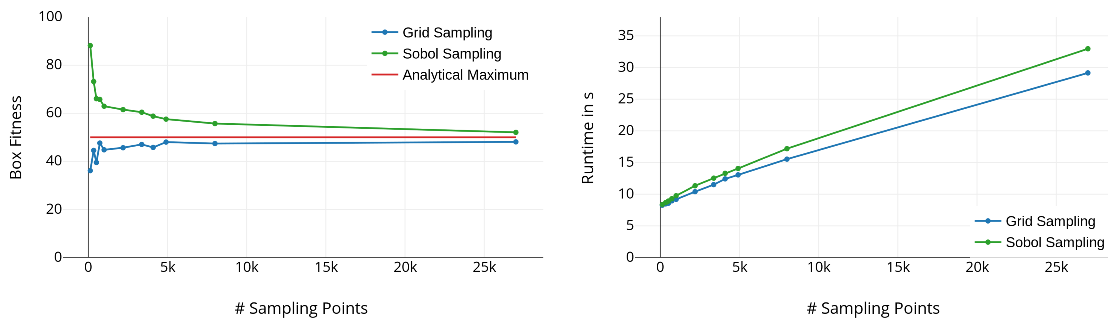


Figure 5.13: Data: 3D Sum Function, **Algorithm**: Anneal Max Box, **Constraints**: $0 \leq f(x) \leq 2.3811$

Random Max Box gives moderate results. Sobol sampling leads to larger boxes than Grid sampling. For Sobol sampling the boxfitness starts with 50 and converges to 35. For Grid sampling the boxfitness starts with 10 and converges to 29. The runtime grows linear with the amount of sampling points (range: 8 - 35 seconds).

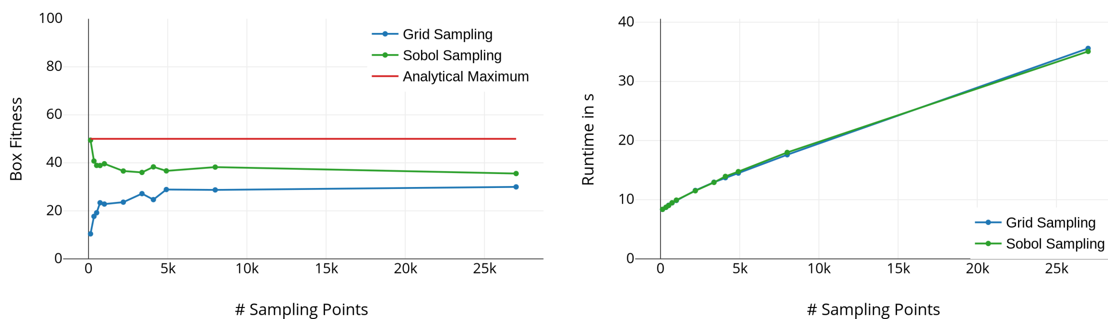


Figure 5.14: Data: 3D Sum Function, **Algorithm**: Random Max Box, **Constraints**: $0 \leq f(x) \leq 2.3811$

x1 Function (1D - 10D)

We define a function which we denote as *x1 function*:

$$f(\mathbf{x}) = x_1,\ x_1 \in [0,1] \tag{5.7}$$

where $\mathbf{x}$ is a vector with arbitrary dimensionality. This function generates data for a supposedly easy test problem to investigate the behaviour of the box maximization algorithms in higher dimensions. As an analytical benchmark for the box fitness, we define again 50. Since the function only depends on the first parameter, a constraint of 0.5 ensures a box fitness of 50.
Sobol sampling allows the distribution of arbitrary amount of sampling points for each dimension. On the contrary, our implementation of Grid sampling needs to have a number of sampling points which is divisible by the number of dimensions. Hence, Sobol sampling is used in dimensional experiments to enable comparability. We use an exponential scale starting with 100 sampling points and increasing with $\sqrt{10}$ in each step until 100 000. Hence, we get the sequence of sampling points: [100, 316, 1000, 3162, 10 000, 31623, 100 000].
Runtime was capped at 15 minutes. If data is missing, this means that the runtime threshold was exceeded. The heuristic algorithms were executed three times in a row to reduce variability of the results.

Exact Max Box Results are shown in Figure 5.15. We see that a solution box is always calculated, except for 100 000 sampling points and more than 4 dimensions where the runtime threshold was exceeded. The runtime seems almost constant in all dimensions until 10 000 sampling points but in reality it slightly increases with every dimension. For example with 1 000 sampling points in one dimension the runtime is 0.034 seconds and with 10 dimensions it is 2.54 seconds. With more sampling points the increase gets steeper.

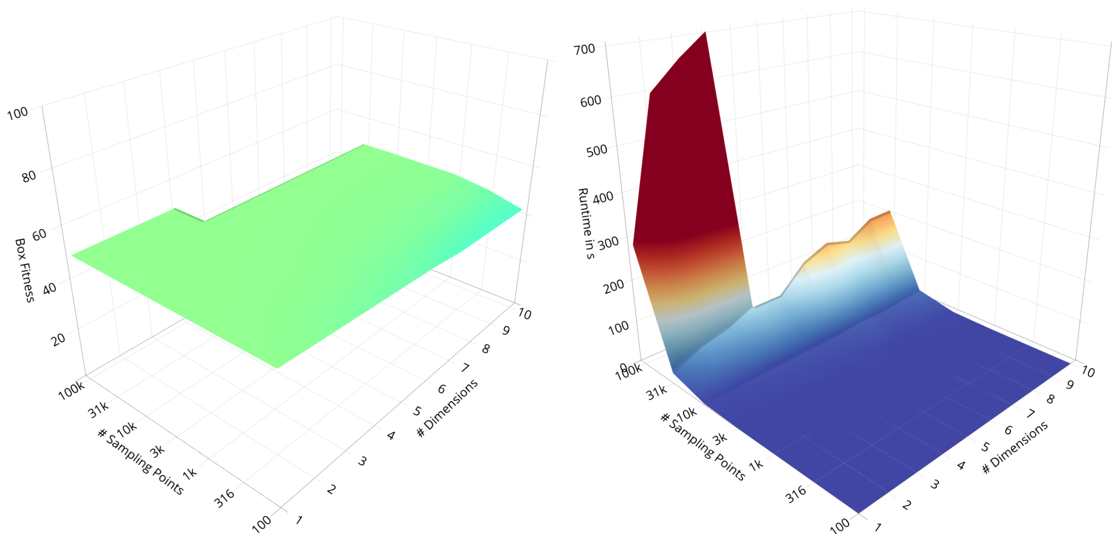


Figure 5.15: Data: x1 Function, **Algorithm**: Exact Max Box, **Constraints**: $0 \leq f(x) \leq 0.5$

Anneal Max Box Results are shown in Figure 5.16. The algorithm almost always finds the analytical maximum except for high dimensions with few sampling points. For example, for eight, nine and ten dimensions the box fitness drop significantly. It seems like the algorithm is not able to navigate out of the region of bad designs, if the dimensions are high and design points are sparse. The runtime is moderate. Even in ten dimension with 100 000 sampling points 5 minutes are barely exceeded.

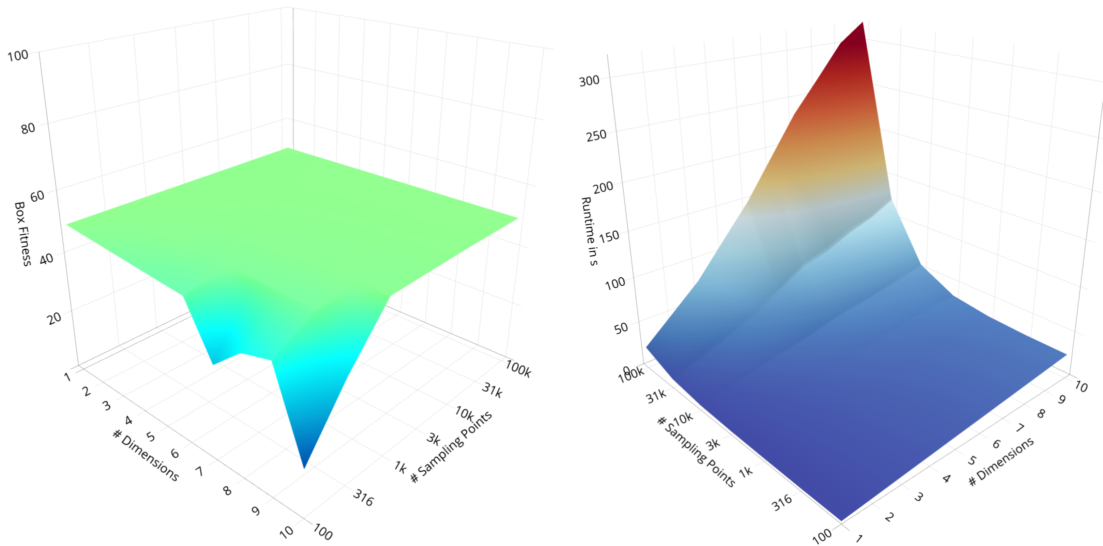


Figure 5.16: Data: x1 Function, **Algorithm**: Anneal Max Box, **Constraints**: $0 \leq f(x) \leq 0.5$. Note that left plot was rotated for better visibility.

Random Max Box Results are shown in Figure 5.17. The algorithm performs very bad for all dimensions greater than 2, independent of the amount of sampling points. This is because in higher dimensions randomly generated boxes occupy less space. In every dimension only a part of each dimension is used to span a solution box. Thus, with every additional dimensions the box gets smaller and the probability that the box comprises design points decreases. The runtime is similar to Anneal Max Box.

Sum Function (1D - 10D)

In this experiment the sum function in the range from one to ten dimensions is used. As an analytical benchmark we chose a volume of 50 for the examples with two and three dimensions. Hence, the shape of the bad region in two dimensions was a triangle (Figure 5.7) and in three dimensions a tetrahedron (Figure 5.11). These geometric objects are called *simplexes* for arbitrary dimensions. In Section A.3, we show that the body of the bad region is still a simplex as long as the largest solution box (good region) has a volume of greater than $100\frac{1}{e} \approx 36.7879441171442322$. Thus, we use this value as an analytical benchmark. Since the volume of those simplexes gets smaller with more dimensions, we wanted to choose the largest possible bad region which still is a simplex. With Equation 5.6 we get $\left[0, p\sqrt[p]{\frac{1}{e}}\right]$ for the constraint on the system response.
The maximum runtime was set to 15 minutes. If areas of the surface plots are missing, this means

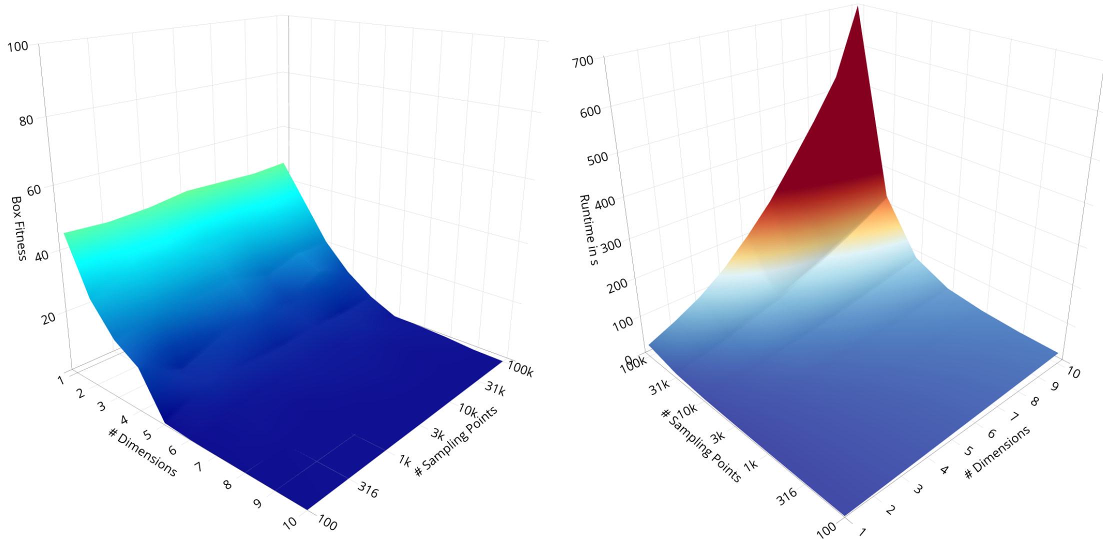


Figure 5.17: Data: x1 Function, **Algorithm**: Random Max Box, **Constraints**: $0 \leq f(x) \leq 0.5$. Note that left plot was rotated for better visibility.

that the algorithm did not terminate in time. The heuristic algorithms were executed three times in a row to reduce variability of the results.

Exact Max Box Results are shown in Figure 5.18. The surface plot is missing some areas because the runtime exceeds the maximum of 15 minutes. In higher dimensions the runtime significantly drops. This seems unintuitve at first because in the prior experiments a higher amount of sampling points led to an increase in runtime. The reason is that there are no or only few bad designs if the amount of dimensions is high and the amount of sampling points is low. This is due to the fact that the simplex of bad designs in higher dimensions almost has no volume. In Section A.3 we show that the volume of this simplex is given by $\frac{1}{p!}(p - f_c^u)^p$. In ten dimensions, for example, the volume of the simplex is ≈ 0.0000001. When 100 000 sampling points are used, the probability to hit such a box with a point can be seen as $0.0000001 \times 100000 = 0.01$. So it is very probable that no bad design point exists in this case. The idea of this example applies analogously to other combinations of dimensions and sampling points. Since the runtime of Exact Max Box algorithm depends on the amount of bad designs, it is probable that very few or no design points in the simplex of bad designs exist. Thus, Exact Max Box terminates fast.

Anneal Max Box Results are shown in Figure 5.19. The results are very good and almost completely resemble the results of Exact Max Box. Note, that Anneal Max Box computes boxes in cases where Exact Max Box is not able to. The runtime increases with more sampling points and more dimensions; getting more steeply with these two parameters.

Random Max Box Results are shown in Figure 5.20. The results are very bad. Almost independently of the amount of sampling points, with each dimension the solution boxes get smaller. The runtime is comparable to the runtime of Anneal Max Box algorithm.

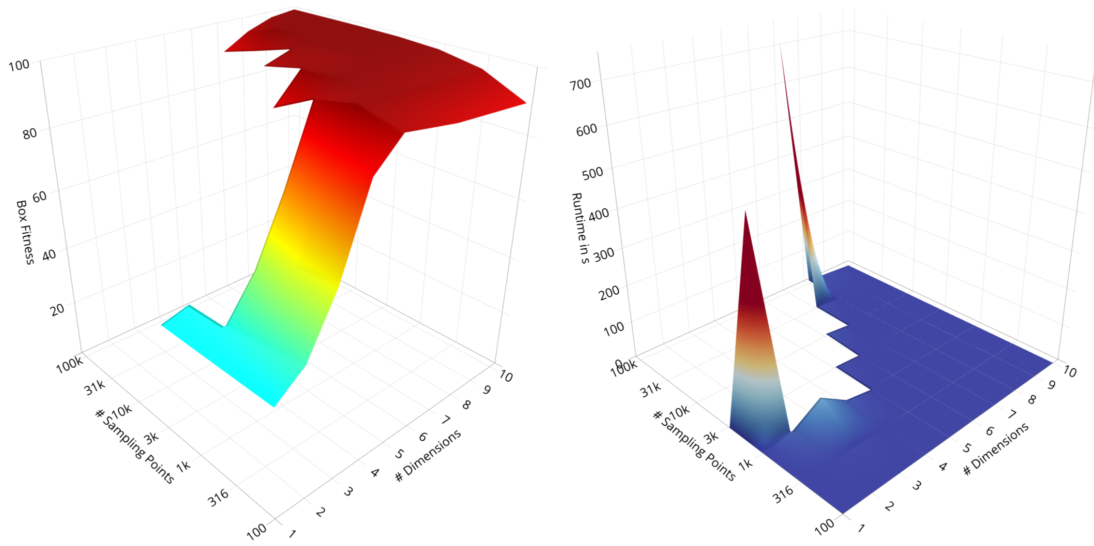


Figure 5.18: Data: Sum Function, **Algorithm**: Exact Max Box, **Constraints**: $0 \leq f(x) \leq p\sqrt[p]{\frac{1}{e}}$.

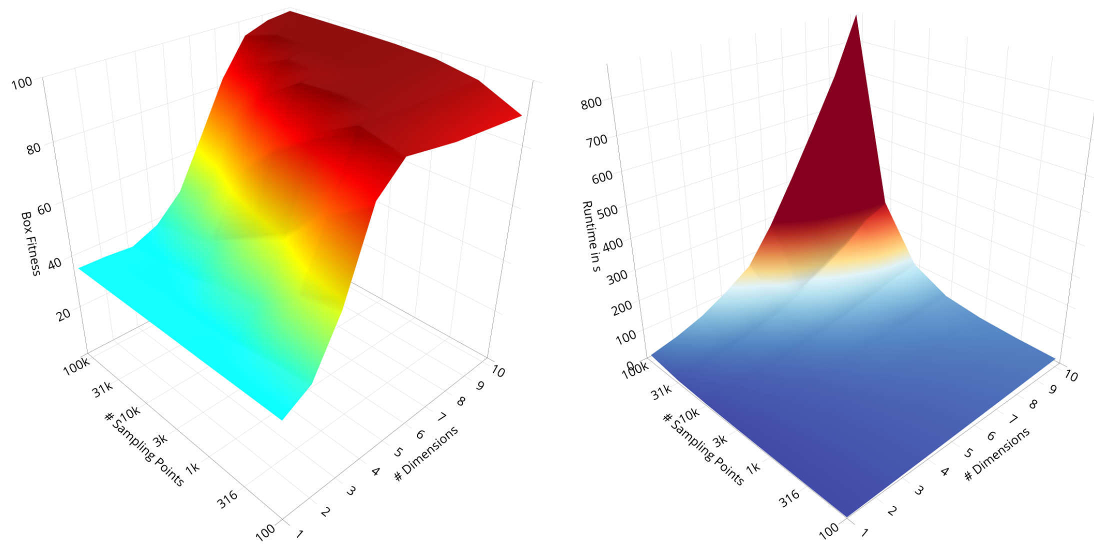


Figure 5.19: Data: Sum Function, **Algorithm**: Anneal Max Box, **Constraints**: $0 \leq f(x) \leq p\sqrt[p]{\frac{1}{e}}$

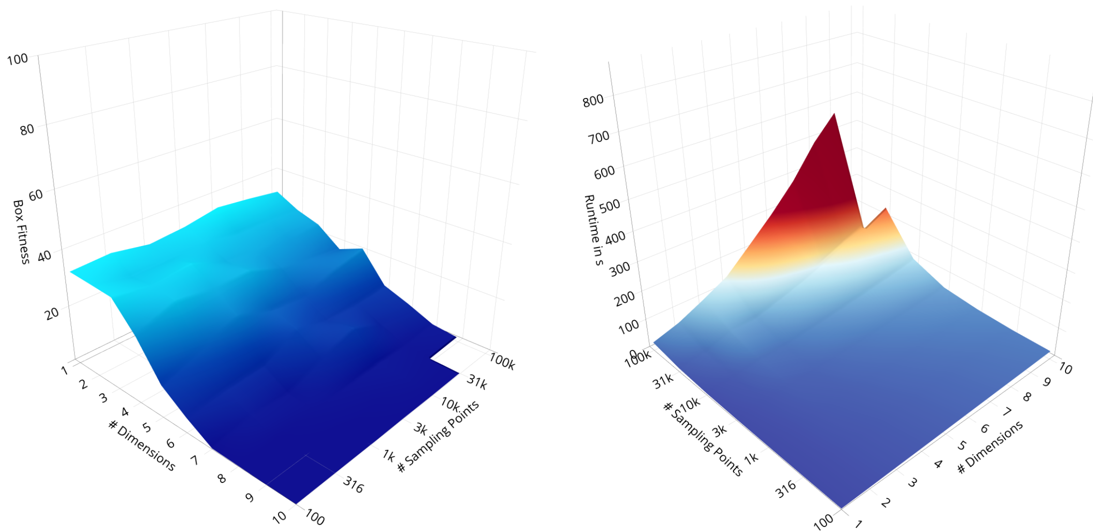


Figure 5.20: Data: Sum Function, **Algorithm**: Random Max Box, **Constraints**: $0 \leq f(x) \leq p\sqrt[p]{\frac{1}{e}}$. Note that left plot was rotated for better visibility.

5.3.2 Nonlinear Problems

Experiments so far concerned linear problems. Since the prototype is applied on nonlinear problems, we run tests with data generated by nonlinear function as well and compare the results with the previous experiments.

Simplified Rosenbrock Function (2D)

In Section 5.2 we already used the Rosenbrock function to generate a clustering problem. In this section we use a simplified version of it to generate a problem where it is possible to calculate the maximum box analytically.

$$f(x, y) = b(y - x^2)^2 \tag{5.8}$$

Again, we set $b = 100$. We refer to this function as the Simplified Rosenbrock function. The system response is constrained with $0 \leq f(x, y) \leq 500$. It generates a two dimensional nonlinear-problem very similar to the one in Figure 5.6. The analytical maximum is 45.5. Runtime threshold was set to 60 minutes.

Exact Max Box Results are shown in Figure 5.21. A similar pattern as with the 2D sum function is visible: data points with Sobol sampling seem to overestimate and Grid sampling seems to underestimate the best possible solution box. Interestingly, the runtime is significantly higher than in the 2D sum function example. Already with 2500 data points and Sobol sampling it took the algorithm 24 minutes to terminate. The algorithm did not terminate within an hour for 4.900

data points or larger with Sobol sampling, hence this data is missing in the diagram. Similarly, the algorithm did not terminate within an hour for 40 000 data points and Grid sampling.

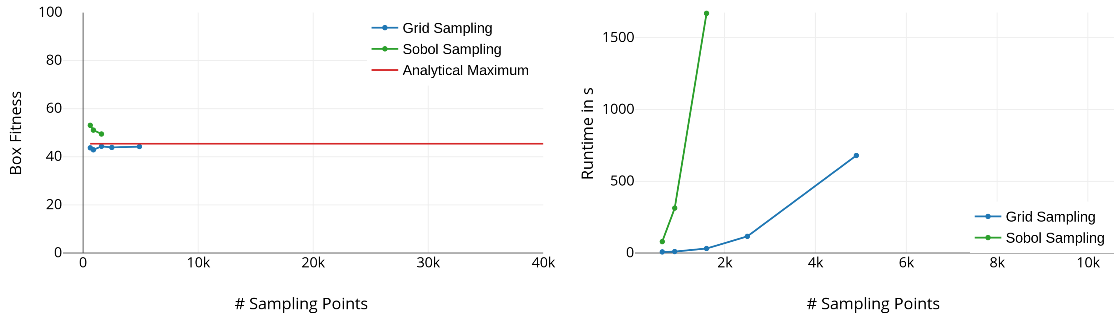


Figure 5.21: Data: Simplified Rosenbrock Function, **Algorithm**: Exact Max Box, **Constraints**: $0 \leq f(x) \leq 500$

Anneal Max Box Results are shown in Figure 5.22. Overall the results are good. The solution boxes are not perfect but converge at 41 (Grid sampling) and 43 (Sobol sampling). Runtime grows linear with sampling points and is slightly higher than in the 2D sum function experiment.

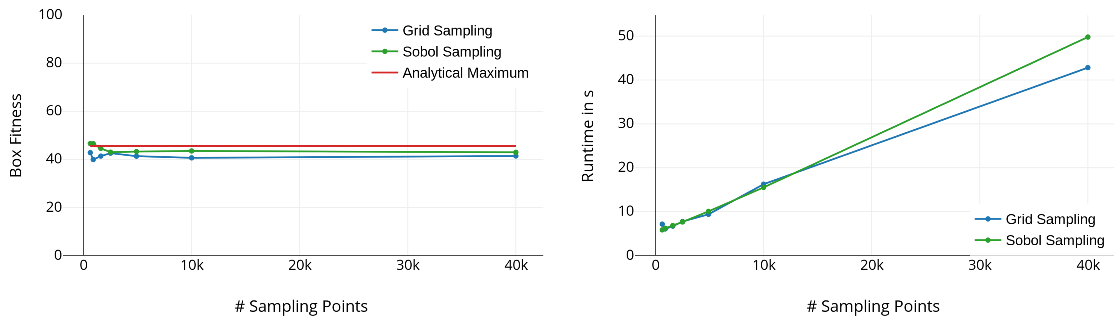


Figure 5.22: Data: Simplified Rosenbrock Function, **Algorithm**: Anneal Max Box, **Constraints**: $0 \leq f(x) \leq 500$

Random Max Box Results are shown in Figure 5.23. Overall the results are moderate. The box fitness converges at 35. Runtime grows linear with sampling points. Interestingly, the runtime almost is double as high as with Anneal Max Box.

Ishigami Function (3D)

A second, more difficult test case for nonlinear problems is created by using the Ishigami function (Equation A.4)

$$f(x, y, z) = \sin(x) + a \sin^2(y) + bz^4 \times \sin(x) \tag{5.9}$$

In our case, we set $a = 7$ and $b = 0.1$. The system response is contstrained by $0 \leq f(x, y) \leq 6.5$. The resulting problem comprises multiple local maximas. This can be seen in the solution space in Figure 5.24. Runtime threshold was set to 60 minutes.

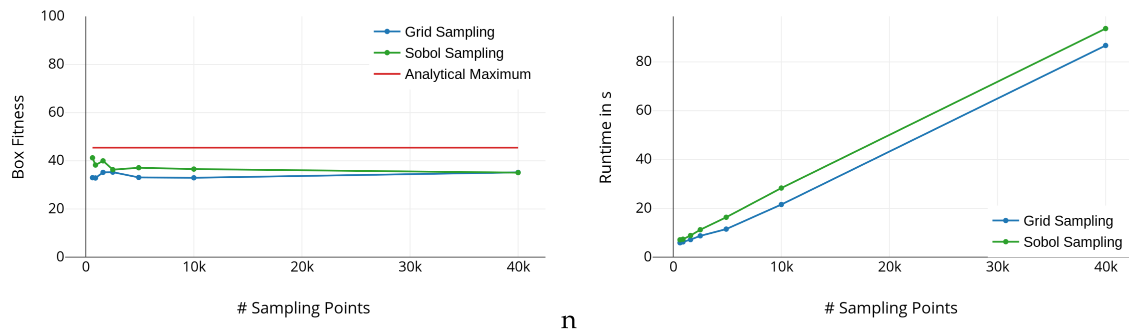


Figure 5.23: Data: Simplified Rosenbrock Function, **Algorithm**: Random Max Box, **Constraints**: $0 \leq f(x) \leq 500$

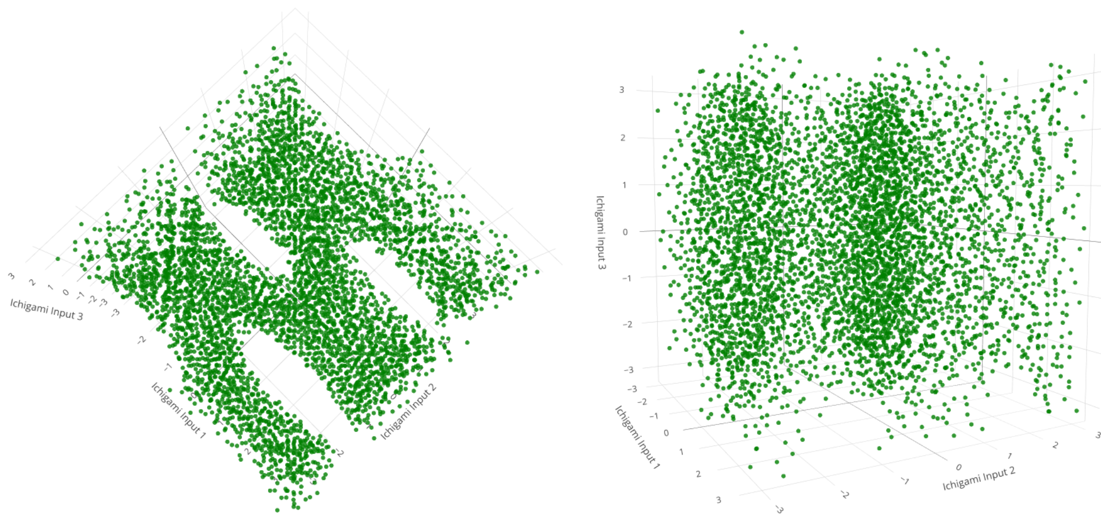


Figure 5.24: Data: Ishigami Function, **Constraints**: $0 \leq f(x) \leq 6.5$, **Sampling Points**: 8 000

Exact Max Box Results are shown in Figure 5.25. Most notably, the runtime of the Exact Max Box in this case increases rapidly with the amount of sampling points. The step from 125 to 512 sampling points leads to an increase from 8 seconds to almost 24 minutes with Sobol sampling. Even though the runtime with Grid sampling does not grows as steep, it still grows exponentially, hitting about 24 minutes 8 000 data points. 27 000 sampling points were tested as well, but the runtime of the algorithms exceeded the threshold. Due to the lack of an analytical benchmark, no absolute statement about the box fitness scores can be made. The box fitness seems to be reasonable, given the visualization and the proportions of the whole design space.

Anneal Max Box Results are shown in Figure 5.26. The results are mixed. With Sobol sampling the boxes converge to a box fitness of 5.3. With Grid sampling the box fitness is only 1.7. In comparison to the solution boxes of the Exact Max Box, those boxes are fine. It has to be noted that the results fluctuate with every trial. The runtime of the algorithm grows linearly independent of the sampling method.

Random Max Box Results are shown in Figure 5.27. Results are moderate. With Sobol sampling the boxes converge to a box fitness of 2.6. With Grid sampling the box fitness is only 2.3. In

comparison to Anneal Max Box, the results for Grid sampling is better. On the other hand the result for Sobol sampling is worse. It has to be noted that the results fluctuate with every trial. The runtime of the algorithm grows linearly independent of the sampling method.

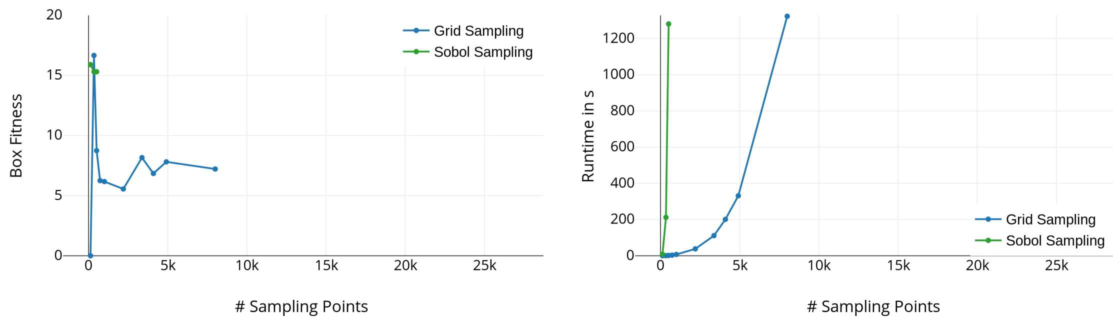


Figure 5.25: Data: Ichigiami Function, **Algorithm**: Exact Max Box, **Constraints**: $0 \leq f(x) \leq 6.5$

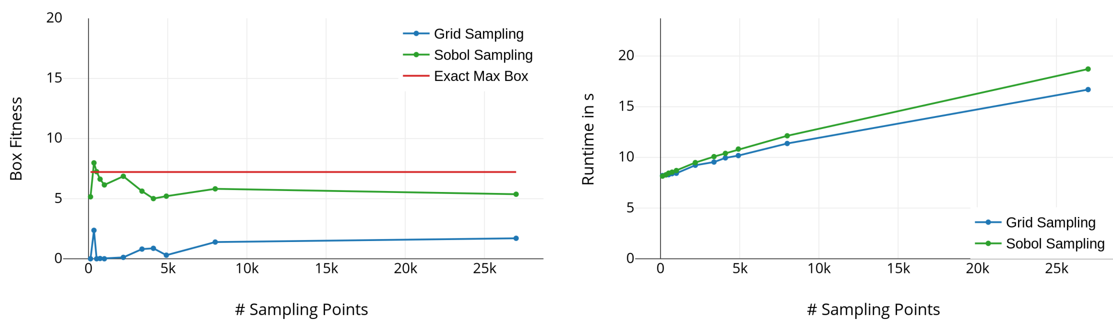


Figure 5.26: Data: Ishigami Function, **Algorithm**: Anneal Max Box, **Constraints**: $0 \leq f(x) \leq 6.5$

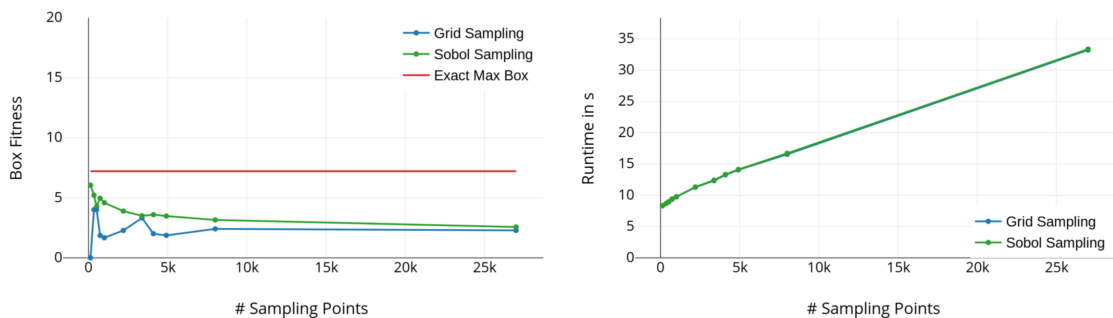


Figure 5.27: Data: Ishigami Function, **Algorithm**: Random Max Box, **Constraints**: $0 \leq f(x) \leq 6.5$

5.3.3 Runtime Difference between Grid and Sobol Sampling with Exact Max Box

A behaviour continually observed in the evaluation is that box maximization problems sampled with Sobol sampling took the Exact Max Box algorithm significantly longer to solve than the same problem with Grid sampling. A hypothesis for this behaviour is the following: Exact Max Box algorithm takes values of bad designs as split values to create subproblems with bounds lower and higher than the split value. When Grid sampling is used, with every split, all bad designs which

have the same value in that dimension will not be considered anymore, because they are not contained in the subproblems. That is not the case with Sobol sampling. Since points are located more arbitrary, it is more probable that the subproblems contain only the negative split point less. This hypothesis is supported by the following experiment. A tweaked Grid sampling was implemented which lets the sampling points in the design dimensions randomly deviate a modicum in each dimension. We refer to this sampling as *Noise Grid sampling*. We see in Figure 5.28 that this significantly increases the runtime. This supports our hypothesis that the property of Grid sampling that many points are located in a line lead to significantly reduced runtime with the Exact Max Box algorithm. However, the runtime with Noise Grid sampling still differs to Sobol sampling by a constant factor. This suggests a relationship between the distribution of design points and runtime: the more evenly a non-grid sampling is distributed the higher the runtime.

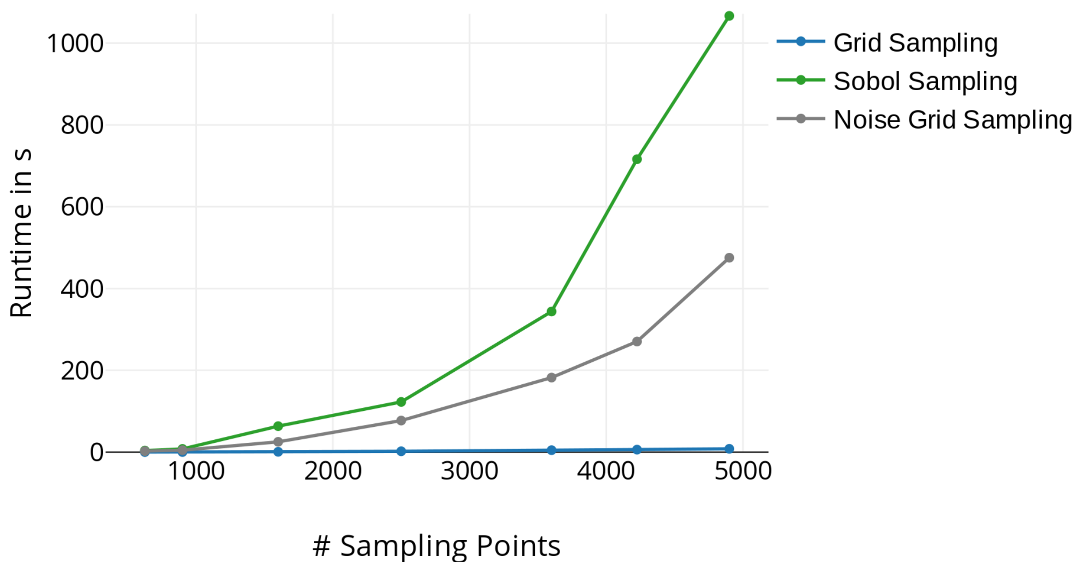


Figure 5.28: Noise Grid Sampling has a significantly higher runtime than Grid sampling when the Exact Max Box algorithm is used.

5.3.4 Summary

Different conclusions are drawn from the box maximization evaluation. First, heuristic algorithms have their justification because the runtime of Exact Max Box grows rapidly with respect to dimensions and amount of bad designs. The use of Sobol sampling always leads to larger solution boxes than the use of Grid sampling. In cases where the analytical benchmark was available, those Sobol boxes were larger then the largest analytically possible box. Grid sampling on the other hand seemed to approach the maximum from below. With a more difficult problem created using the Ishigami function this distinction seems not to hold anymore.
With regard to the algorithms, Anneal Max Box performed significantly better than Random Max Box (baseline) in our linear test problems from one to ten dimensions. Also, it performs better for nonlinear problems with only one global maximum (Simplified Rosenbrock Function). In the case of difficult nonlinear problems with multiple local maximas (Section 5.3.2) Anneal Max Box performs better than Random Max Box when Sobol sampling is used. In this case, if Grid Sampling is used, it is the other way round. Practically, Random Max Box is not usable in problems

with more then three design parameter dimensions.
In general, there is no rule on how to choose the amount of sampling points. It depends heavily on the box maximization problem which is to be solved. Naturally, the more sampling points are available, the more accurately the largest solution box can be approximated. Even though more sampling points lead to more accurate representation of the underlying function, more designs also mean signficant increases in runtime in all tested algorithms. With easy linear test problems we saw the box fitness converges with 900 points in two dimensions and 25 000 in three dimensions. For more complex problems, those numbers need to be higher. 50 000 desisgns seem to be already to few designs for more than five dimensions.

6 END-TO-END EVALUATION

In the end-to-end evaluation we use two real datasets to test the software prototype: the SCALE dataset and the US-NCAP dataset. In this chapter the whole pipeline (Figure 2.7) is tested.

6.1 SCALE DATASET

The SCALE dataset is a normalized and anonymized dataset of real crash tests. It consists of 63 samples with twelve design parameters and two system responses. All columns are normalized to the interval $[0, 1]$. Since the system response Y varies between 0 and 1, we set $0 \leq Y \leq 0.5$ as the constraint on Y. In this test the second system response Y2 is left out. Further information about the SCALE dataset can be found in the appendix (Subsection A.1.1).

A dataset with two design parameter dimensions allows for the creation of plots to visualize the results. For reasons of comprehensibilty, we create such a dataset with two design parameters (C, I) and a system response (Y). We chose C and I because in previous investigations we saw that the dataset is very sensitive to those two design parameters. We refer to this subset of the full dataset as *2D Scale*. Nevertheless, the test is also conducted with the complete dataset, i.e. twelve design parameters and system response Y. We refer to this dataset as *12D Scale*.

6.1.1 Outlier Detection and Model Quality

For each dataset a polynomial regression model is created. The models were created with one, two and three degrees. The prototype automatically chooses the model with the best R^2 score on the test set. As stated in Section 5.1, the SCALE dataset contains one known outlier. The table below compares the R^2 scores for the polynomial regression models created with and without this outlier. Model generation was repeated with different test and training sets until the third digit after the decimal point converged. Values were rounded to two digits after the decimal point.

	Outlier		No Outlier	
Dataset	R^2 Test	R^2 Training	R^2 Test	R^2 Training
2D Scale	0.46	0.64	0.58	0.73
12D Scale	0.57	0.79	0.59	0.85

In the case of two design parameters we see an increase of 0.12 on the predictive power on the test set. On the training set the R^2 score increases by 0.09. In the case of 12 design parameters, the increase in predictive power is smaller but still significant both on the test set (0.02) and training set (0.06).
We found that the 12D Scale model created with the augmented dataset was essentially an useless model even though the R^2 scores look good. The predicitions were very bad and either large negative or positive numbers; not as expected in the range from 0 to 1. At the time of writing we could not explain the reason for this behaviour. To circumvent the problem a neural network model with Tensorflow was created instead and used in the following experiments.
In our tests it came to know that model generation with Tensorflow took significantly longer than with polynomial regression. In the case of 12D Scale it took polynomial regression 0.45 seconds to finish with three trials (degrees one, two, three). To train a neural network with random search and three trials (two times 300 epochs and one time 100 epochs) took about 38 seconds. Per trial polynomial regression took only 0.15 seconds to finish while Tensorflow needed 12.94 seconds.

6.1.2 Data Augmentation

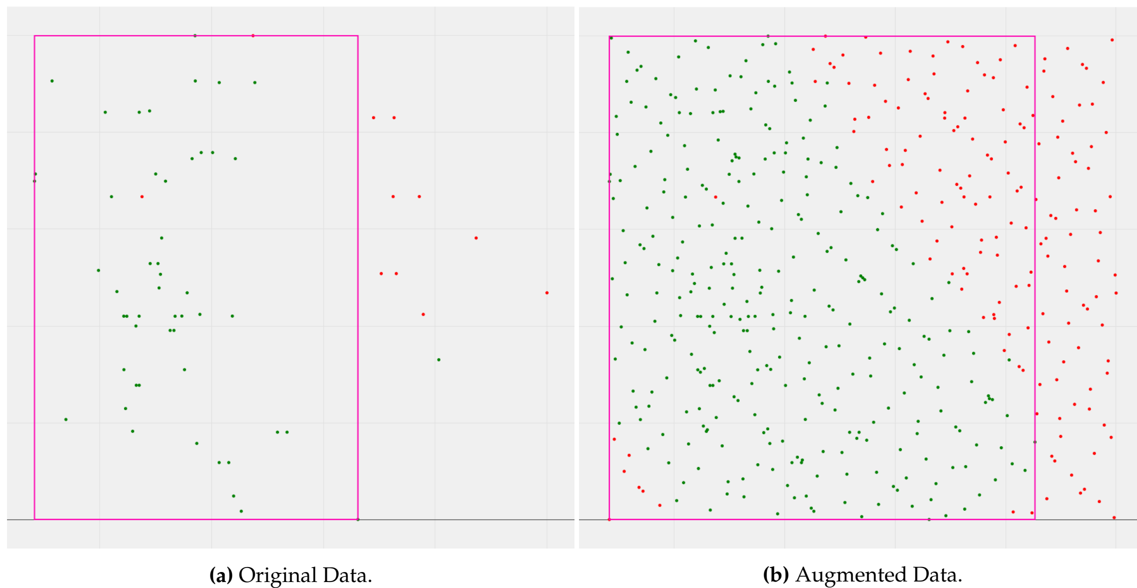
In Figure 6.1a the original design space for the 2D SCALE dataset is visualized. In Figure 6.1b the design space is augmented by 400 additional designs using the polynomial regression model created in the previous step. Likewise, data augmentation was performed on the 12D SCALE dataset.

(a) Original data. **(b)** Data augmented by 400 Sobol sampled design points.

Figure 6.1: Two dimensional subset of the SCALE dataset. **Constraint**: $0 \leq Y \leq 0.5$

6.1.3 Clustering

In the 2D SCALE dataset clusters are found properly, both in the case of real data (Figure 6.2a) and with 400 additional designs (Figure 6.2b). In the case of 12D SCALE dataset without data augmentation, a small cluster with a volume of 0 is found. This is because 63 designs for a 12 dimensional design space is a really sparse design space. As this cluster is too small for additional tests, we only consider the augmented 12D SCALE dataset in the following box maximization step.



(a) Original Data. **(b)** Augmented Data.

Figure 6.2: One cluster is found on the 2D SCALE dataset both with and without data augmentation.

6.1.4 Box Maximization

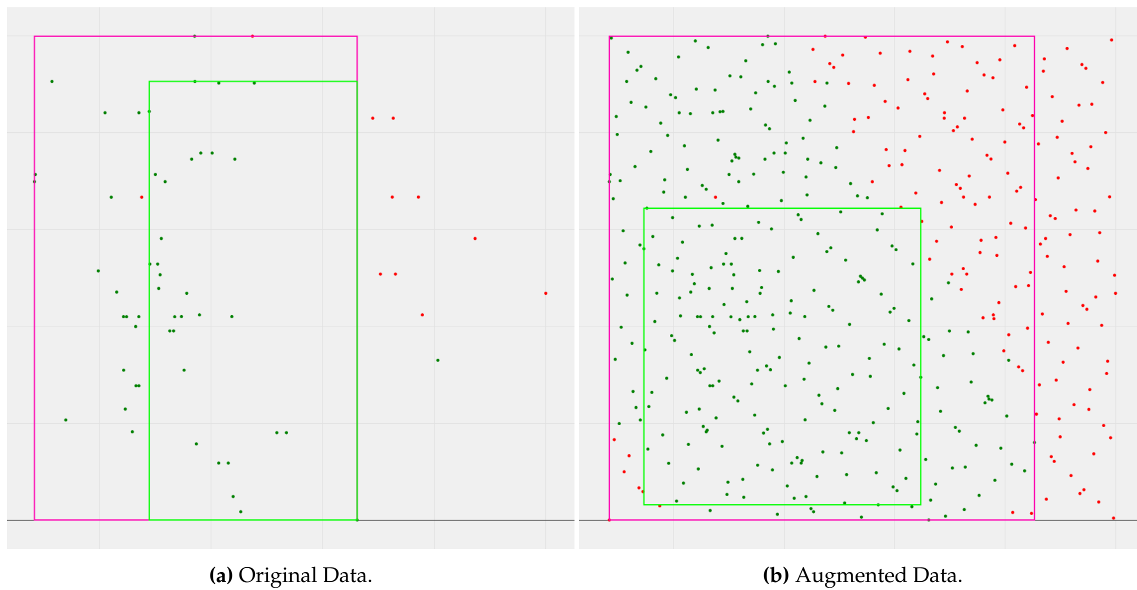
Due to the variance of the heuristic box maximization algorithms, Anneal Max Box and Random Max Box are exectued five times in a row for each test case. We average the results to get a more robust box fitness. Nonetheless, we show the highest and lowest value of the results so that one can get a sense of how this value can vary.

2D Scale

With two design parameters Exact Max Box can be used to calculate solution boxes. The results on the original data are shown in Figure 6.3a. Likewise, the results on the augmented data are shown in Figure 6.3b. The results of the box maximization algorithms can be found in the table below.

# Algorithm	Real Data Lowest	Real Data Highest	Real Data ∅	Augmented Data Lowest	Augmented Data Highest	Augmented Data ∅
Exact Max Box	36.9	36.9	36.9	33.5	33.5	33.5
Anneal Max Box	34.3	36.9	36.4	26.7	30	28.7
Random Max Box	17.1	25.6	19.7	22.7	30	26.3

The largest possible solution boxes for the augmented design space is smaller than the box fitness for the original design space. Interestingly, Anneal Max Box often calculates the maximum box in the original design space. Naturally, Exact Max Box performs slightly better than Anneal Max Box. Random Max Box finds solution boxes which are on average smaller than the boxes Anneal Max Box finds.



(a) Original Data. **(b)** Augmented Data.

Figure 6.3: One cluster is found on the 2D SCALE dataset both with and without data augmentation.

12D Scale

The results of the box maximization algorithm on the 12D SCALE dataset can be found in the table below.

# Algorithm	Augmented Data Lowest	Augmented Data Highest	Augmented Data ∅
Exact Max Box	n/a	n/a	n/a
Anneal Max Box	22	36	28.7
Random Max Box	0	0	0

In twelve dimensions Exact Max Box can not calculate a solution box in reasonable time. Anneal Max Box calculates solution boxes which on average fill 28.7% of the design space. However, Random Max Box solution boxes have a volume of 0. This means that in at least one dimension an interval only consisted out of one value.

6.2 US-NCAP DATASET

The US-NCAP dateset contains data about a front crash of the NCAP Ford Taurus model [26]. It consists of 27 design parameters and twelve system responses. Further information can be found in the appendix (Subsection A.1.2).

Two additional subsets of the US-NCAP dataset are used. One of those subsets should consist of three design parameters to be able to plot the results. The other subset should comprise ten design parameters. Again, we refer to the datasets as *3D US-NCAP*, *10D US-NCAP* and *27D US-NCAP*. The design parameters for the datasets are choosen on the basis of the sensitivity analysis which was conducted on the US-NCAP dataset by Reuter et al. [26]. The analysis was conducted on the system response *max_ intrusion_ firewall_ left*. Therefore, *max_ intrusion_ firewall_ left* is used as a system response and the other system response are left out in this test. The values for this response vary between 25.14 and 343.02. We set $0 \leq max_intrusion_firewall_left \leq 200$ as a constraint. For the dataset 3D US-NCAP *mat_ rO*, *mat_ rI* and *rail_ I* are choosen as design parameters. For the test with 10D US-NCAP additionally *rail_ O*, *mreinf*, *subfr*, *subfr_ l*, *arm_ top*, *arm_ bot* and *tie_ bar* are used. These design parameters are choosen because they show a bigger effect on *max_ intrusion_ firewall_ left* over all used sensitivity analysis methods in comparison to other design parameters.

6.2.1 Outlier Detection and Model Quality

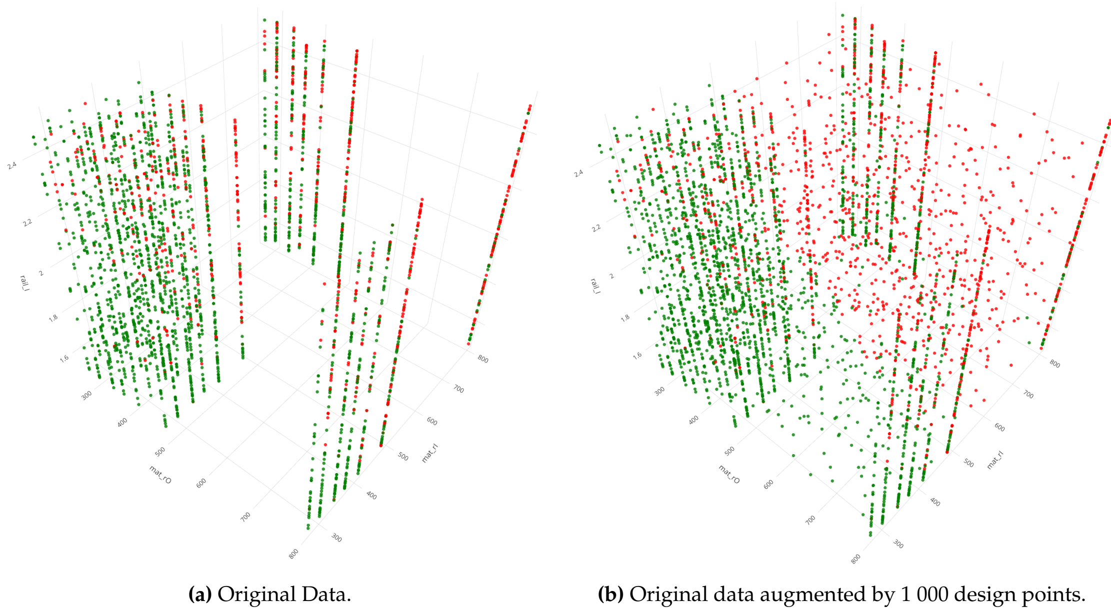
Polynomial regression was used to create models. Again, one, two and three degrees is used and the best model is choosen. As we saw in Section 5.1 there are 118 known outliers in the dataset which can be found by outlier detection. The R^2 scores both on the test and training set can be found in the table below. Model generation was repeated with different training and test sets and averaged until the third digit after the decimal point converged. Values were rounded to two digits after the decimal point.

	Outlier		No Outlier	
# Design Parameters	R^2 Test	R^2 Training	R^2 Test	R^2 Training
3D US-NCAP	0.00	0.00	0.41	0.44
10D US-NCAP	-0.01	0.00	0.53	0.56
27D US-NCAP	0.02	0.05	0.72	0.80

The resulting R2-Scores between the models generated with and without outliers differ significantly. Models generated with outliers have a R^2 scores of almost 0. Those models effectively have no predictive power. On the other hand, if outliers are removed, the resulting models are reasonable. The more design parameter dimensions are available (and with that: the more information), the better the resulting models.

6.2.2 Data Augmentation

In Figure 6.4a we see that the US-NCAP dataset shows visible gaps between designs. This underlines the importance of data augmentation. Therefore, in this part we focus on augmented data instead of only real data. In Figure 6.4b the dataset with data augmentation is shown. Sobol sampling was used to create 1 000 additional design points.



(a) Original Data. **(b)** Original data augmented by 1 000 design points.

Figure 6.4: 3D SCALE dataset with and without data augmentation.
Constraint: $0 \leq max_intrusion_firewall_left \leq 200$

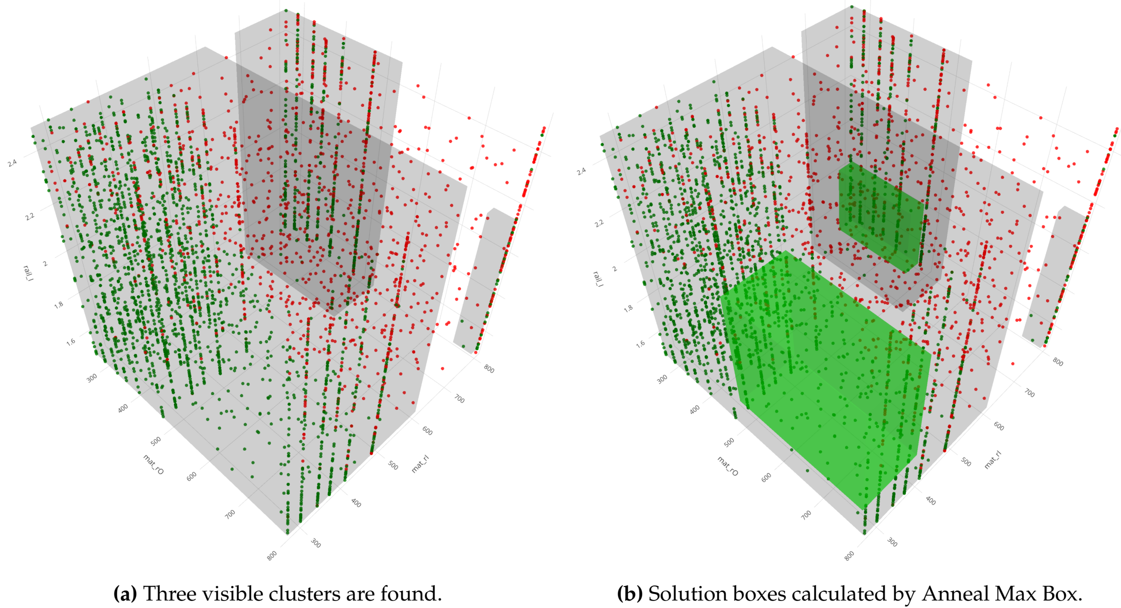
6.2.3 Clustering

In the subset with three design parameters five clusters are found. Because two of those clusters contain only two designs, we want to leave them out. We filter out all clusters with less than six data points. The remaining three clusters are visualized in Figure 6.5a. In 10D US-NCAP dataset one big cluster is found. In the full dataset 60 clusters are found. Again, only one cluster is reasonably large. The other 59 clusters contain mostly only two to five design points. They are filtered out. In the next step we use the large cluster for the box maximization of the 27D US-NCAP dataset.

6.2.4 Box Maximization

3D US-NCAP

Again, because of the variance, Anneal Max Box and Random Max Box are exectued five times in a row for each test case. Results are averaged to get a more robust box fitness. Nonetheless, the highest and lowest value of the results are shown so that one can get a sense of the variance.



(a) Three visible clusters are found. **(b)** Solution boxes calculated by Anneal Max Box.

Figure 6.5: 3D SCALE dataset augmented with 1 000 design points.
Constraint: $0 \leq max_intrusion_firewall_left \leq 200$

Solution boxes calculated by Anneal Max Box are visualized in Figure 6.5b. In the table below box fitnesses of the solution boxes for the three algorithms is shown.

# Algorithm	Augmented Data		
	Lowest	Highest	∅
Exact Max Box	n/a	n/a	n/a
Anneal Max Box	6.5	10.36	8.42
Random Max Box	1.46	2.55	2.23

Exact Max Box could not provide a solution box in reasonable time. In a time frame of 12 hours it did not terminate. Anneal Max Box gave good results. Random Max Box gave moderate results. In summary, Anneal Max Box performed best.

10D US-NCAP

Again, Exact Max Box could not provide a solution in a reasonable amount of time. Anneal Max Box calculated better solution boxes than Random Max Box on average. However, the variance of the box fitness seems to increase with more design parameter dimensions.

# Algorithm	Augmented Data		
	Lowest	Highest	∅
Exact Max Box	n/a	n/a	n/a
Anneal Max Box	0.77	4.24	2.28
Random Max Box	0	2.57	0.54

27D US-NCAP

When all design parameters are used, Exact Max Box can not provide a solution. Anneal Max Box provides solution boxes which vary significantly in each trial. Random Max Box only generates useless boxes: a box fitness of $-\infty$ signifies that the provided boxes do not contain any design point. Not in a single trial Random Max Box found a box better than $-\infty$.

	Augmented Data		
# Algorithm	Lowest	Highest	∅
Exact Max Box	n/a	n/a	n/a
Anneal Max Box	1.42×10^{-5}	32570.92×10^{-5}	10766.9×10^{-5}
Random Max Box	$-\infty$	$-\infty$	$-\infty$

6.3 SUMMARY

The end-to-end evaluation yields many insights. We see that outlier removal can significantly improve the predictive power of models. The generation of models with polynomial regression is significantly faster than with Tensorflow. At the same time the resulting models are similar in terms of their R^2 score on test and training set. However, a model with a high R^2 score can still be a bad model. As we saw in the case of 12D SCALE dataset the polynomial regression model had a high R^2 score both on training and test set. Still, the predictions of the model were very bad. In those cases neural network models should be used instead.
Clustering works independent of the number of design parameters. However, no general statement about the clustering quality can be made due to the lack of a benchmark. The clusters made sense visually in two and three dimensions. Sometimes a large amount of tiny clusters are found. They can be removed by filtering.
Practically, Exact Max Box can not be used on Sobol sampled augmented datasets with more than two dimensions. Anneal Max Box is usable in high dimensions, but the box fitness of the solutions have a high variance. Due to the lack of a benchmark, no definite statement about the quality of the solution boxes can be made. Random Max Box has the tendency to produces bad solution boxes in high dimensions. This is consistent with the results in Section 5.3.1 where we saw that solution boxes get smaller with higher dimensions when Random Max Box is used. This trend continues with dimensions greater than ten. In 27 dimensions the box fitness hits $-\infty$. Practically, Random Max Box should not be used on datasets with more than three desgin dimensions.
An important observation is that even though the solution boxes seem small because the volume is small, the permissible intervals can still be large.

7 DISCUSSION

It can be summarized that the software prototype is able to determine permissible intervals for arbitrary dimensions. On easy test problems in arbitrary dimensions the Anneal Max Box algorithm finds good solution boxes. In certain low-dimensional cases even the Exact Max Box algorithm is applicable. Furthermore, in higher dimensions Anneal Max Box is able to provide solution boxes, even though the solutions vary significantly between trials. The methods for outlier detection showed that they can contribute in significantly improving model quality. In the tested cases the clustering algorithm finds clusters and provides multiple solutions boxes for the engineer. Visualizations of the dataset in different steps of the pipeline help to interpret the results, make problems comprehensible and support in interpreting the dataset.
However, the approach of the prototype still has its limitations. The approach in this thesis assumes to create fixed amount of design points for a problem before a box maximization process is started. This sampling process tries to approximate the underlying function of the dataset. In high dimensions a substantial amount of sampling points is needed to approximate that function. Unfortunately, more sampling points mean also a significantly higher runtime for the algorithms. Even worse, in some cases the amount of sampling points needed are too high to calculate a solution with the heuristic algorithms. For example, if we assume that 50 sampling points per dimension are needed to approximate a highly nonlinear problem, in six dimensions already more than 15 billion sampling points are needed. The problem can also be visualized from another perspective: if an engineer expects all of his permissible intervals to comprise 30 % of each design parameter range in a ten dimensional design space, this leads to a volume of $0.3^{10} = 0.000005905$. If the design space is sampled evenly, the probabilty to hit this volume with a sampling point can be seen as 0.000005905. The amount of needed sampling points necessary to sample this space is at least $\frac{1}{0.000005905} \approx 169348$. Thus, about 170 000 sampling points are needed so that it is probable that just *one* point is located in the permissible space. At least two points are needed to span a solution box and two points have a very low probability to be able to span the largest box so that the full 30 % intervals in each dimension can be found. Thus, even more than 340 000 points in this setup can not guarentee be sufficient to find the desired solution box.
This problem can be described as a *resolution problem* in high dimensions. It partially stems from the fact that our approach is conservative which means that solution boxes need to have supporting designs on their border. This problem would not emerge if the algorithms would allow protruding edges in empty spaces to be considered as permissible. However, if the algorithm

would not be conservative, engineers can not have that much confidence in the resulting solution boxes.

The resolution problem is a typical problem arising from high dimensions. The well-known phenomena that certain problems emerge from high dimensions is called the *curse of dimensionality*. Therefore, there is great interest in reducing the amount of dimensions without impairing the modeled system. One approach is sensitivity analysis which aims to determine the most significant variables of a function. Implementing sensitivity analysis in the prototype could improve the results if the amount of dimensions of some design parameters turn out not to contribute much or anything to the system response. This is why sensitivity analysis is proposed as useful extension to the existing prototype.

In general, engineers can only have confidence in the provided permissible intervals insofar as the generated model of the dataset is of good quality. Thus, it is of utmost importance that the generated models have high R^2 scores or low least-squares errors, respectively. A problem is that the generated model heavily depends on the provided dataset. If there are not enough data points, it could be helpful to indicate to the engineer that he is about to generate a model which can not have a good prediction quality. If he still uses a bad model for determination of permissible intervals, the prototype should warn the user that those intervals should be used with caution. With respect to the user experience, the prototype would benefit from hints guiding the engineer in the process of outlier detection, model generation, design space sampling, clustering and box maximization.

Furthermore, the Anneal Max Box algorithm has potential for improvement. In high dimensions and with difficult low-dimensional problems, solution boxes have a high variance. From this it can be concluded that the algorithm is not deterministic in those cases. To make it deterministic either the amount of trials can be increased or the algorithm in itself has to be revised with trial and error on relevant test problems. Furthermore, there exists the possibility to implement heuristic algorithm with a different metaheuristic such as an evolutionary / genetic programming. The exact procedure may depend on the practical problems and the specific application of the software prototype.

In terms of sampling, we saw that Grid sampling in general is not applicable because no arbitrary amounts of sampling points can be generated. On the other hand, Random sampling leads to bad distribution of sampling points. This is why Sobol sampling is the favoured approach for sampling. We see no reason why a different sampling method should improve the results. An idea worth investigating instead would be an adaptive / intelligent sampling of the design space. Since often the design space is more filled in certain areas than in others, some areas need more sampling than others. Runtime could be saved, if those sampling points were only created in the sparse areas of the design space.

8 CONCLUSION

In this thesis, the engineers' problem of optimizing designs while still satisfying overall design goals was presented. This problem leads to a need to support engineers in the design process by determining permissible intervals of design parameters where the system response fulfills a predefined constraint. It was shown that permissible intervals need to be independent of each other to be easily usable by the engineer. On the basis of the work of Zimmermann et al., we showed that this problem is an optimization problem where the largest multidimensional box in solution space has to be calculated. The aim of this thesis was to develope a software solution to determine those independent permissible intervals.
In the beginning and on the basis of related work a concept for the prototype was created. The concept comprises a step for data preprocessing and the subsequent determining of permissible intervals. Data preprocessing contains the removal of outliers via outlier detection methods and data augmentation which itself consists out of model generation and design space sampling. On the preprocessed data permissible intervals are determined by a clustering step and a box maximization step. The concept served as a basis to find and implement methods to solve the problems of each step of the pipeline. For outlier detection proximity-based methods such as k nearest neighbour and DBSCAN were presented and implemented. As an example for high-dimensional outlier detection, isolation forest was explained and integrated. For data augmentation the topics model generation and sampling methods were investigated. On that basis two model generation methods were implemented: polynomial regression and artificial neural networks. For sampling methods, Grid sampling and Sobol sampling are used. For clustering the algorithm DBSCAN was reused. As a solution for the box maximization algorithms an exact algorithm by Eckstein et al. was presented and implemented. Exact Max Box always provides a best solution to the box maximization problem. Because Eckstein et al. showed that their algorithm is NP-hard, the topic of metaheuristics was explored and the Anneal Max Box algorithm, which is inspired by simulated annealing, was implemented. As a naive baseline solution Random Max Box algorithm was implemented. All important steps of the workflow were augmented by meaningful visualizations.
In a component-wise evaluation outlier detection methods, clustering and box maximization steps were evaluated. Anneal Max Box algorithm determined good solution boxes for easy test problems in arbitrary dimensions. For more difficult test problems with multiple local maximas the algorithm provided solutions which varied between trials. In an end-to-end evaluation the prototype was tested completely on two real datasets - the SCALE dataset and the US-NCAP

dataset. The evaluation showed that in general the approach works for arbitrary dimensions if Anneal Max Box is used. However, we saw that solutions of Anneal Max Box vary more significantly between trials in higher dimensions. In general, it came to know that the approach suffers from the curse of dimensionality. We showed that the approach has a *resolution problem* which means that in higher dimensions, depending on the problem, too many design points are needed. To overcome this, a sensitivity analysis was proposed to reduce the amount of irrelevant dimensions. Further future work is the implementation of an intelligent / adaptive sampling as well as the improvement of Anneal Max Box. Furthermore, a new algorithm could be implemented leveraging other metaheurtics such as genetic or evolutionary programming.

A APPENDIX

A.1 REAL DATASETS

Real datasets are datasets which were either generated by real world measurements or CAE simulations.

A.1.1 SCALE Dataset

The SCALE dataset is a normalized dataset of real crash tests. Due to the nature of real experiments, the data in this dataset is noisy. It consists of 63 samples with fourteen parameters each. Twelve parameters are available for design and two parameters are system responses. It contains one known outlier. A scatterplot matrix of the dataset can be found in Figure A.1.

A.1.2 US-NCAP Dataset

The US-NCAP dateset is data about a front crash of the NCAP Ford Taurus model [26]. It consists of 2972 samples with 39 parameters each. 27 parameters are available for design and twelve parameters are system responses. It contains 118 known outliers. The known outliers all have values of 2×10^{30} on the system response parameters. At the end of writing this thesis, it came to know that those values are the typical output values of the simulation program LS-DYNA in the case that certain calculations failed.

A.2 SYNTHETIC DATA

In the thesis, data is generated on-the-fly using functions for evaluation purposes. In this section we describe all functions used for test data generation, as well as datasets which were generated.

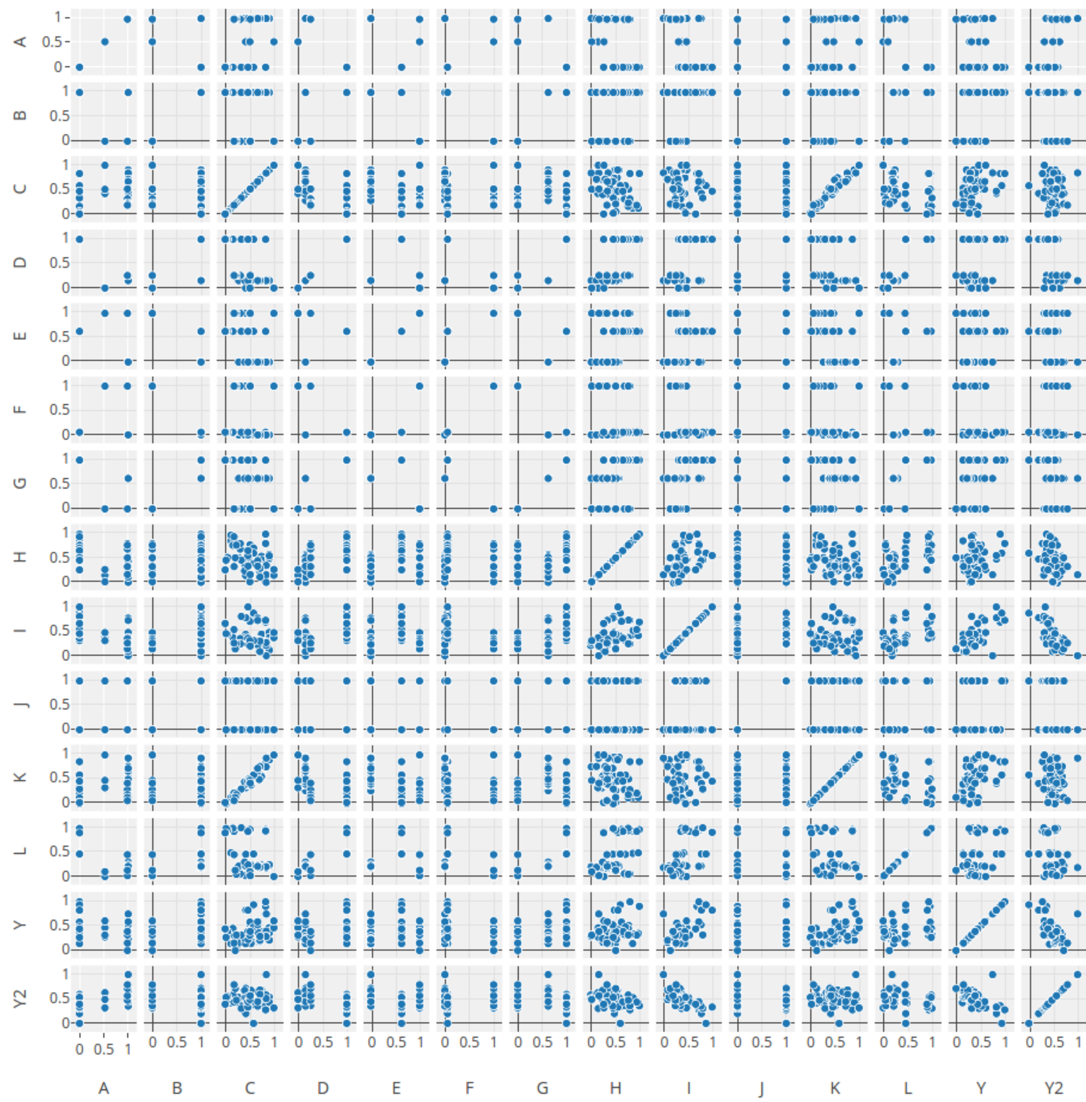


Figure A.1: Scatterplot matrix of the SCALE dataset.

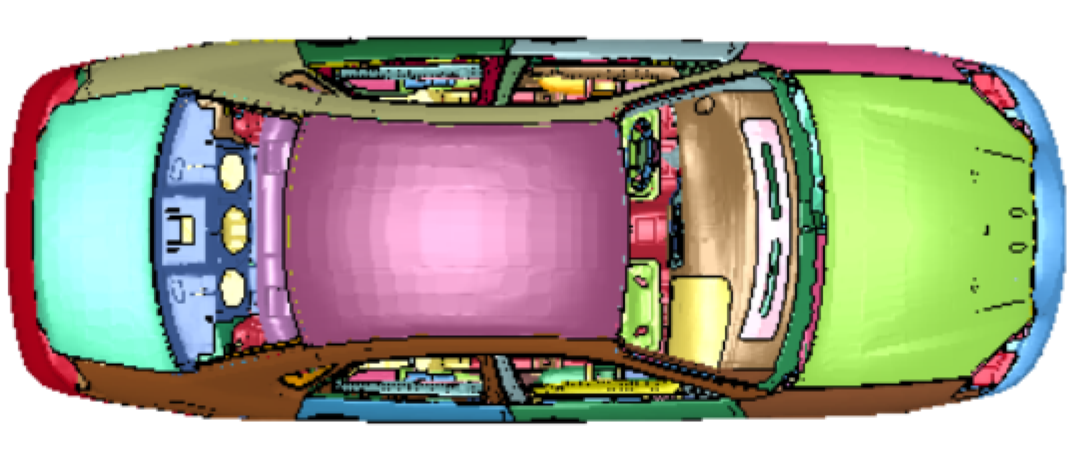


Figure A.2: Ford Taurus model front crash test, 56.6 km/h [26].

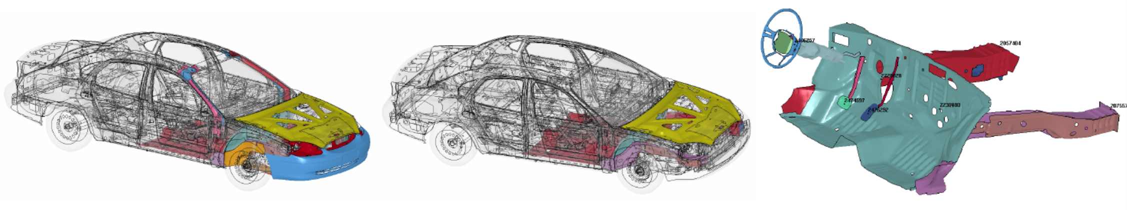


Figure A.3: US-NCAP dataset contains 27 design parameters and twelve system response variables [26]
.

A.2.1 Functions for Data Generation

This section summarizes all functions used for data generation. To generate data, the design space is filled with data points. In this thesis two sampling methods are being used:

- Sobol sampling
- Grid sampling

Also, the number of data points is variable and depends on the usage context. Those two parameters for data generation are mentioned if necessary.

Identity Function

The identity function maps a value or vector to itself.

$$Id(x) = x \tag{A.1}$$

Rosenbrock Function

The Rosenbrock function is non-convex function used as a performance test problem for algorithms in optimization. [27]. The function is given by:

$$f(x,y) = (a-x)^2 + b(y-x^2)^2,\ x,y \in [-1,3] \tag{A.2}$$

In our case, we set $a = 1$ and $b = 100$.

Simplified Rosenbrock Function

From the Rosenbrock function we derive a simplified function which we refer to as *Simplified Rosenbrock Function*.

$$f(x, y) = b(y - x^2)^2,\ x, y \in [-1, 3] \tag{A.3}$$

In our case, we set $b = 100$.

Ishigami Function

$$f(x, y, z) = \sin(x) + a\sin^2(y) + bz^4 \times \sin(x),\ x, y, z \in [-\pi, \pi] \tag{A.4}$$

In our case, we set $a = 7$ and $b = 0.1$.

Sum Function

We define a *sum function*:

$$f(\mathbf{x}) = \sum_{i=1}^{p} x_i, x_i \in [0, 1], p \in \mathbb{N}_{\neq 0} \tag{A.5}$$

with n being the number of dimensions of the vector $\mathbf{x}$.

A.2.2 Generated Datasets

Identity Dataset

The identity dataset was generated using identity function (A.2.1). The Identity dataset has 200 samples and two columns. The first column contains data points created with Sobol sampling in the range [0, 10]. Values in the second column are given by the identity function, thus it contains same values as in the first column. Manually, 9 outliers were created in the dataset. The outlier values of $Id(x)$ deviate in different magnitudes.

A.3 PROOFS

In this section we prove four statements made in the context of the box maximization test problem generated with the sum function (Section A.2.1). Recall that the sum function is defined on the p-dimensional unit cube with $x_i \in [0, 1]$. By B_{max} we denote the largest box fitting inside the space of good designs defined by the relation $\sum_i x_i \leq f_c^u \in [0, p]$:

1. The box B_{max} is a cube.
2. Prescribing the box fitness of B_{max} to a value μ is equivalent to requiring $f_c^u = p\sqrt[p]{\frac{\mu}{100}}$.
3. If $\text{Vol}(B_{max}) \geq V_{min} = \frac{1}{e}$, the set of bad designs $R = \{x | \sum_{i=1}^{p} x_i > f_c^u\}$ is a simplex.
4. If R is a simplex, we have $\text{Vol}(R) = \frac{1}{p!}(p - f_c^u)^p$.

Here $e \approx 2.71$ denotes Euler's number.

Proof of 1: Let $\mathrm{Vol}(B)$ a volume of an arbitrary box B. Then $\mathrm{Vol}(B) = L_1 \times \ldots \times L_p$ where $L_1, \ldots, L_p$ are the length of the edges of B. With the well-known *inequality of arithmetic and geometric means* we get

$$L_1 \times \ldots \times L_p \leq \left(\frac{L_1 + \ldots + L_p}{p}\right)^p$$

The equality only holds if $L_1 = \ldots = L_p$ which is equivalent of the box being a cube. □

Proof of 2: With the considerations of Proof 1 and because $L_1 + \ldots + L_p = f_c^u$, we get

$$\mathrm{Vol}(B) = \left(\frac{f_c^u}{p}\right)^p$$

Thus, for a given volume $\mathrm{Vol}(B)$ and $\mu = 100\mathrm{Vol}(B)$ for a box with only good designs

$$f_c^u = p\sqrt[p]{\mathrm{Vol}(B)} = p\sqrt[p]{\frac{\mu}{100}} \tag{A.6}$$

□

Proof of 3: We are looking for a value V_{min} for which $\mathrm{Vol}(B_{max}) \geq V_{min}$ implies that R is a simplex. In two dimensions R is a simplex (triangle) if $f_c^u \geq 1$. In three dimensions it is a simplex (tetrahedron) if $f_c^u \geq 2$. In general, this is the case if and only if $f_c^u \geq p - 1$. Thus, $V_{simplex} = (\frac{p-1}{p})^p = (1 - \frac{1}{p})^p$. With the well-known fact that $\lim_{x \to \infty}((1 - \frac{1}{p})^p)$ converges to $\frac{1}{e}$ from below, we get the desired assertion. □

Proof of 4: $\mathrm{Vol}(R) = \mathrm{Vol}(\{x | \sum_{i=1}^p x_i > f_c^u\}) = \mathrm{Vol}(\{x | \sum_{i=1}^p x_i \leq p - f_c^u\})$. With the formula for the volume of a standard simplex $\mathrm{Vol}(\{x | \sum_{i=1}^p x_i \leq p\}) = \frac{1}{p!}$ and that the volume of a body in a p-dimensional space gets scaled by a scale factor s with s^p, we get: $\mathrm{Vol}(R) = \frac{1}{p!}(p - f_c^u)^p$. □

BIBLIOGRAPHY

[1] Charu C. Aggarwal. *Outlier Analysis*. Springer Publishing Company, Incorporated, 2nd edition, 2016.

[2] I.A. Antonov and V.M. Saleev. An economic method of computing lp_τ-sequences. *USSR Computational Mathematics and Mathematical Physics*, 19(1):252 – 256, 1979.

[3] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, February 2012.

[4] Chire. Dbscan illustration. https://commons.wikimedia.org/wiki/File:DBSCAN-Illustration.svg (Last retrieved: 2020-02-23).

[5] J.G. van der Corput. Verteilungsfunktion. i. mitt. *Proceedings of the Koninklijke Akademie van Wetenschappen te Amsterdam*, (38):813–821, 1935.

[6] Josef Dick and Friedrich Pillichshammer. Digital nets and sequences: Discrepancy theory and quasi-monte carlo integration. *Digital Nets and Sequences: Discrepancy Theory and Quasi-Monte Carlo Integration*, 01 2010.

[7] Jonathan Eckstein, Peter L. Hammer, Ying Liu, Mikhail Nediak, and Bruno Simeone. The maximum box problem and its application to data analysis. *Comput. Optim. Appl.*, 23(3):285–298, December 2002.

[8] Henri Faure. Discrépance de suites associées à un système de numération (en dimension s). *Acta Arithmetica*, 41(4):337–351, 1982.

[9] Lavinia Graff, Helmut Harbrecht, and Markus Zimmermann. On the computation of solution spaces in high dimensions. *Structural and Multidisciplinary Optimization*, 54:811–829, 2016.

[10] J.H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numer. Math.*, (2):84–90, 1960.

[11] Julian Haluska. *isolation-forest* npm package. https://www.npmjs.com/package/isolation-forest (Last retrieved: 2020-02-24).

[12] Helmut Harbrecht, Dennis Tröndle, and Markus Zimmermann. *Approximating solution spaces as a product of polygons*. Preprints Fachbereich Mathematik, 2019.

[13] D. Hawkins. *Identification of outliers*. Monographs on applied probability and statistics. Chapman and Hall, London, 1980.

[14] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automated Machine Learning: Methods, Systems, Challenges*. Springer, 2018. In press, available at http://automl.org/book.

[15] Anthony Karpathy. Neural networks part 1. http://cs231n.github.io/neural-networks-1/ (Last retrieved: 2020-02-21), 2015.

[16] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[17] Lukasz Krawczyk. *density-clustering* npm package. https://www.npmjs.com/package/density-clustering (Last retrieved: 2020-03-10).

[18] David Kriesel. A brief introduction to neural networks. http://www.dkriesel.com/_media/science/neuronalenetze-en-zeta2-2col-dkrieselcom.pdf (Last retrieved: 2020-02-21).

[19] F. T. Liu, K. M. Ting, and Z. Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, Dec 2008.

[20] J. Sander M. Ester, H.-P. Kriegel and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. *Data Mining and Knowledge Discovery*, pages 226–231, 1996.

[21] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, pages 281–297, Berkeley, Calif., 1967. University of California Press.

[22] Michael Nielsen. Deep learning and neural networks, 2015.

[23] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, second edition, 2006.

[24] Tor Nordqvist. *sobol* npm package. https://www.npmjs.com/package/sobol (Last retrieved: 2020-02-24).

[25] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Longman Publishing Co., Inc., USA, 1984.

[26] Uwe Reuter, Zeeshan Mehmood, Clemens Gebhardt, Martin Liebscher, Heiner Müllerschön, and Ingolf G. Lepenies. Using ls-opt for meta-model based global sensitivity analysis. 01 2011.

[27] H. H. Rosenbrock. An Automatic Method for Finding the Greatest or Least Value of a Function. *The Computer Journal*, 3(3):175–184, 01 1960.

[28] I. M. Sobol. On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, 7(4):86 – 112, 1967.

[29] K. Sörensen, M. Sevaux, and Glover F. A history of metaheuristics, 2017.

[30] Kenneth Sörensen. Metaheuristics – the metaphor exposed. *International Transactions of Operations Research*, In Press, 01 2013.

[31] Reinhard Stahn. *regression-multivariate-polynomial* npm package. https://www.npmjs.com/package/regression-multivariate-polynomial (Last retrieved: 2020-02-24).

[32] Gerhard Steber. *Reifegradbasierter Gestaltungsprozess für Strukturbauteile im Fahrzeugbau*. 09 2019.

[33] Markus Zimmermann and Johannes Edler von Hoessle. Computing solution spaces for robust design. *International Journal for Numerical Methods in Engineering*, 94(3):290–307, 2013.