Computation of Generalized Solution Spaces

Inauguraldissertation
zur
Erlangung der Würde eines Doktors der Philosophie
vorgelegt der
Philosophisch-Naturwissenschaftlichen Fakultät
der Universität Basel

von
Dennis Tröndle
aus
Deutschland

Basel, 2020

Genehmigt von der Philosophisch-Naturwissenschaftlichen Fakultät
auf Antrag von

Prof. Dr. Helmut Harbrecht
Prof. Dr. Rolf Krause

Basel, den 18.02.2020

Prof. Dr. Martin Spiess
Dekan

Abstract

Solution spaces are applied in distributed design processes. They enable an independent and robust development of the components of a target design. A solution space is a region which contains only good designs and lies in a potentially high-dimensional design space. By finding an appropriate solution space, the design processes for individual components can be decoupled from each other. This increases the efficiency of the overall design process and saves valuable resources.

An established method to find solution spaces is the box optimization algorithm. It provides solution spaces which are products of intervals and take on the shape of a high-dimensional, axis-parallel box. We review this method and give a detailed account of how different parameter settings affect the outcome of the algorithm.

The box optimization algorithm yields sometimes intervals that are too small. To this end, we develop the rotated box optimization algorithm. It couples specific pairs of components and rotates the corresponding box. Thus, it is able to find boxes with a larger volume and increases the amount of available good designs.

An algorithm which might yield even larger solution spaces is the polytope optimization algorithm. Instead of trying to find boxes which are as large as possible, it maximizes the volume of polytopes. Because polytopes have a much more flexible shape than boxes, this gives rise to larger solution spaces compared to the previous algorithms. However, the algorithm is more complex and requires additional steps to handle the polytopes.

We compare these algorithms by applying them to several high-dimensional optimization problems. Our results show that, indeed, the polytope optimization algorithm yields the solution spaces with the largest volume.

Acknowledgements

I want to express my deepest gratitude to my supervisor Prof. Dr. Helmut Harbrecht for taking me on as a PhD student and for guiding me through my doctoral studies. He has been a constant source of support and counsel, and taught me many mathematical and non-mathematical skills. Our frequent travels to and from Munich have always been unique and often turned out to be more eventful than we expected.
I would like to thank Prof. Dr. Rolf Krause for taking the time to be the co-referee of my PhD thesis.
Many thanks go to Prof. Dr. Markus Zimmermann who pitched the initial draft of this project when he worked fo the BMW Group. He sparked many new ideas and discussions, and helped us understand the engineer's point of view. The BMW Group has my thanks for the funding I received from them and their hospitality at the Forschungs- und Innovationszentrum (FIZ) in Munich. I would also like to thank Stefan Erschen and Marc-Eric Vogt from the BMW Group for showing me the FIZ and for their input to our discussions.
A lot of thanks go to the current and former members of our research group: Rahel Brügger, Monica Bugeanu, Jürgen Dölz, Ilja Kalmykov, Manuela Moor, Michael Multerer, Marc Schmidlin and Peter Zaspel. I had a wonderful time with all of them and enjoyed all the z'vieris we had and all the activities we did together. They have become great friends to me and I am very happy that they endured all the long board games I persuaded them to play. Special thanks go to Rahel Brügger and Marc Schmidlin who proofread this thesis. Additionally, I thank the people in the administration of the department, who do a great job and keep everything running smoothly.
Of course, a special mention goes to my friends outside of the research group. They have always been there for me and were ready to provide distractions whenever I needed one.
Finally, I am truly grateful to my whole family for their continuous support during my time as a PhD student. Especially, I would like to thank my parents Beate and Johannes for their love and for raising me to believe in myself and my capabilities.

Contents

Chapter 1

Introduction

1.1 Motivation

In today's world, there is a large variety of complex technical products: everyday devices like cars, smartphones and TVs, highly specialized machines such as MRI scanners, satellites or assembly line robots, large-scale projects, e.g. skyscrapers, hydroelectric power plants or particle accelerators, and less tangible products as, for example, software or computer networks. Due to their complexity, a single person can hardly understand every detail of these products. Instead, many people work together to design, develop and create them. Thus, an efficient and well-structured design process is necessary to organize the work flow and to guide everyone who is involved.

Naturally, uncertainty is present in many design processes, especially in the early stages. One type of uncertainty is based on the lack of knowledge, the so-called *epistemic uncertainty*. It governs distributed design processes, where individual teams of engineers are responsible for different components of the final design. In the beginning of the design process, the final characteristics of a component and how it interacts with other components is still uncertain. Also, during the design process, it may turn out that a design is technically impossible to realize, too expensive or in conflict with other requirements. In this case, some components have to be updated to make the design feasible again. Thus, the engineering teams have to iterate over the respective designs multiple times, moving the design in different directions, until they finally agree on a design that fulfills all requirements. At that point, the design is not changed anymore and, in consequence, the uncertainty is removed from the design process. This is a traditional *point based* design process, compare Figure 1.1 for an illustration.

If carried out in this way, the design process requires a lot of time and resources. Thus, more advanced design philosophies are sought. In *concurrent engineering*, all components of a design and processes related to it, like manufacturing, distribution and disposal, are engineered simultaneously. However, since the final design is still unclear in the beginning, new uncertainties arise in concurrent engineering, as its coordination is non-trivial and people from multiple fields have to work together (see [81, 82]). Another approach is *set-based design*, compare [54, 71, 72]. Before committing to a design, the design teams identify sets of feasible designs, e.g. by

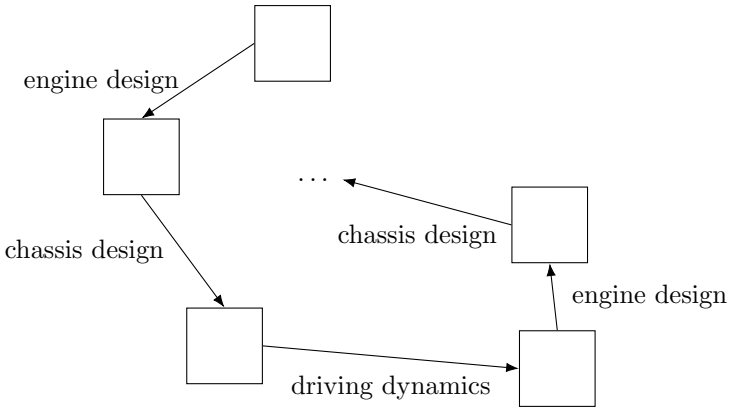

Figure 1.1: A vehicle design being improved iteratively.

giving intervals for parameters that represent the design. Design teams restrict themselves to work only with values derived from these sets. This enables a sustainable design process, where sound decisions can be made due to a large amount of available data. The disadvantage is that a lot of effort has to be made in the early design phase to identify the sets of feasible designs and to keep design teams properly up to date on these sets. A third design philosophy is *set-based concurrent engineering* (see [72]), which combines set-based design and concurrent engineering into one technique. A design process modelled after this philosophy yields good results very quickly. However, it also inherits the disadvantages of set-based design and concurrent engineering and requires experienced designers that are familiar with both philosophies.

As the design methodologies mentioned above require as much information as possible, the lack of knowledge about the final design, i.e., epistemic uncertainty, reduces their efficiency. Thus, we wish to minimize the impact of epistemic uncertainty on the design process. In the case where there is no uncertainty, the goal of computation would be to find the optimal design with respect to an objective function f out of a space of admissible designs Ω_{ds}. This would lead to solving an optimization problem

$$f(\boldsymbol{x}) \to \min_{\boldsymbol{x}\in\Omega_{\text{ds}}}, \tag{1.1.1}$$

where $\boldsymbol{x} := (x_1, \ldots, x_d)$ describes a single design. Each component of the design $\boldsymbol{x} \in \Omega_{\text{ds}}$ is described by one or more of the *design variables* $x_1, \ldots, x_d$. The *design space* Ω_{ds} is in general given as a product of intervals, i.e.

$$\Omega_{\text{ds}} := \prod_{i=1}^{d} [\alpha_i, \beta_i] .$$

We emphasize that f is typically a very involved function for which an analytic form might not be available. For this reason, (1.1.1) is a so-called *black-box optimization problem*.

There are many popular methods to solve this kind of problem. If the evaluation of f is cheap, applicable techniques are

- gradient-based techniques, e.g. gradient descent [13, 87] and quasi-Newton methods [87] such as the DFP method [19, 30] and especially the BFGS method [11, 29, 38, 68],
- direct search techniques, e.g. coordinate descent [83] and the Nelder-Mead method [59],
- metaheuristic techniques, e.g. genetic algorithms [37, 46, 56], differential evolution [1, 73], particle swarm optimization [22, 70] and variable neighborhood search [33, 57].

In the case that the evaluation of f is costly, one may first wish to approximate it with a surrogate model [9, 64]. It is then possible to apply interpolation by radial-basis functions [10], kriging, also known as Gaussian process regression [52, 55, 66], or support vector machines [75] to find a solution for (1.1.1).

All of the aforementioned methods eliminate the epistemic uncertainty by delivering an optimal solution with respect to the objective function f, which means that every team of engineers will know exactly what the specifications of their components are. However, this solution might not be *robust*. Small deviations from the optimal design might lead to a catastrophic loss of quality. These may be caused by the variability of certain materials used to build the design, or because the design is exposed to certain unpredictable conditions, like extreme weather. In general, they cannot be controlled directly and represent an uncertainty that is induced by randomness, a so-called *aleatoric uncertainty* (see [20]).

1.2 An Illustrative Example

An example problem where aleatoric uncertainty affects the design process has been given in [85]. In accordance with the guidelines of the US New Car Assessment Program (see [79]), a car with a speed of 56 km/h is crashed against a rigid barrier. After the crash, the deformation of the front of the car is observed. We assume that it affects only two sections of the car, see Figure 1.2. Section 1 is the front of the car up to the headlights, section 2 goes from the end of the headlights to the center of the front wheels. These two sections have yet to be designed. We do not know what the optimal design is, but we are able to deduce it in the following.

During the crash, two constant deformation forces F_1 and F_2 act on sections 1 and 2, respectively. The total deformation of the front is given by the deformation measure $d \leq d_c$, where d_c is the highest possible deformation. d is given by the deformation measures $d_1 \leq d_{1c}$ and $d_2 \leq d_{2c}$ for sections 1 and 2, where $d := d_1 + d_2$. As the deformation should only happen in the front structure, the rest of the car is assumed to be rigid. The impact energy is given by $E_I := \frac{1}{2}mv_0^2$, where m is the total mass of the car and v_0 the impact velocity. The maximum deformation energy of the first section $F_1 d_{1c}$ is assumed to be smaller than E_I, which means that the first section cannot fully absorb the impact and the second section will also be deformed.

The *design goals* for this problem are the following:

1. The deceleration a of the interior of the vehicle, where the passengers sit, has to be below a critical threshold a_c.

2. The deformation of the front should happen in an *ordered* fashion, meaning that section 1, which is in the front, should be deformed before section 2.

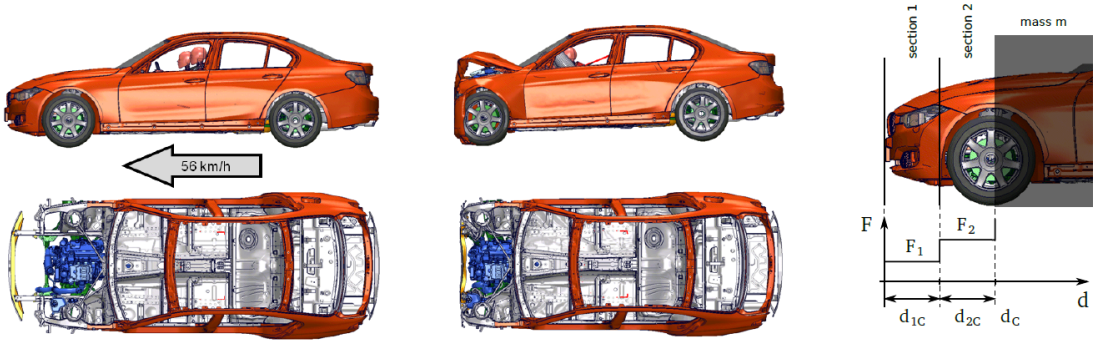

Figure 1.2: Car crash (left and middle) and the two sections in the front of the car (right). Image taken from [85]. Copyright ©2013 John Wiley & Sons, Ltd.

Design goal 1 is fulfilled if the impact energy is fully absorbed before $d = d_c$, i.e., if $E_I \leq F_1 d_{1c} + F_2 d_{2c}$. The maximum deceleration is then given by $a := F_2/m$. Otherwise, it is arbitrarily large. Design goal 2 is fulfilled if $F_1 \leq F_2$. These requirements yield the objective function

$$f(F_1, F_2) := \begin{cases} 1, & \text{if } E_I > F_1 d_{1c} + F_2 d_{2c}, \\ 1, & \text{if } F_1 > F_2, \\ \left(F_2/m - a_c\right)/a_c, & \text{otherwise,} \end{cases}$$

which measures the quality of the design. In the notation from (1.1.1), the *design variables* are given as $x_1 := F_1$ and $x_2 := F_2$. A design (F_1, F_2) with

$$f(F_1, F_2) \leq 0$$

satisfies both design goals and is considered to be good. A visualization of f can be found in Figure 1.3.

Obviously, the minimum of f is

$$(F_1, F_2) = \left(\frac{m v_0^2}{2 d_c}, \frac{m v_0^2}{2 d_c}\right).$$

However, this optimum is not robust. It lies on the corner of the region of good designs and is close to designs that violate one or both of the design goals, as can be seen in Figure 1.3. If F_1 is slightly larger, then $F_1 > F_2$ and the deformation no longer happens in the correct order. Vice versa, if F_1 is slightly smaller, then $E_I > F_1 d_{1c} + F_2 d_{2c}$, meaning that the impact energy cannot be fully absorbed by the two sections and the other regions of the car are deformed. These small deviations from the optimum may be introduced when the actual components are produced. For instance, due to technical limitations or cost restrictions, certain components might

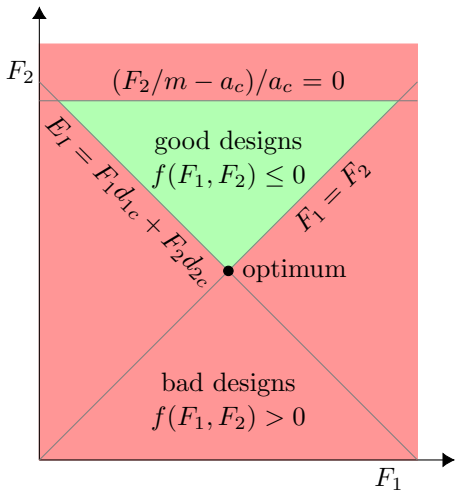

Figure 1.3: A visualization of the objective function f. The region of good designs, where $f(F_1, F_2) \leq 0$, is drawn in green.

only be fabricated with reduced accuracy. Because it is then uncertain whether a car manufactured with these components actually fulfills all design goals, the optimum is hardly a good choice for a robust design. This in turn makes the identification of the optimum rather pointless.

The question that subsequently arises from this example problem is the following: *How can one eliminate epistemic uncertainty while keeping aleatoric uncertainty under control?*

Thus, we will give an overview of methods that handle epistemic and aleatoric uncertainty in the subsequent section.

1.3 Overview of Methods

For general optimization problems, it is possible to account for aleatoric uncertainty by increasing the robustness of the solution. This could be done by *robust design optimization*, *reliability-based design optimization* or *sensitivity analysis*, see [3, 5, 12, 16, 18, 78, 84] for example. These methods require certain assumptions on the problem under consideration. For robust design optimization and reliability-based design optimization, the variability of the design variables has to be known, which may be expressed by probability density functions. Sensitivity analysis requires the derivatives of certain constraints on the design variables. Unfortunately, this type of data is typically not available for the uncertainty related to distributed design processes.

In addition to treating uncertainty appropriately, the following challenges arise in the context of distributed design processes:

- The design variables $x_1, \ldots, x_d$ are coupled with each other, i.e., they simultaneously affect the overall system performance.

- The evaluation of the objective function f is expensive. Therefore, it is mandatory to keep the number of function evaluations small.
- f is a black-box function. It is possibly noisy and no information about the gradient is available. Hence, classical optimization techniques cannot be applied.
- There is a large number of design variables, so Ω_{ds} is high-dimensional.

We present a few methods that are able to deal with these challenges. They all follow the philosophy of set-based design and yield a wide range of possible designs rather than a single optimal design. This overview is by no means complete. It is instead intended to give the reader an impression of the variety of techniques and combinations thereof that are applied in set-based design.

Rules Extraction

In [31], a method is described that generates rules for sets of good designs from *linear support vector machines* (SVM, see also [7]). A linear SVM identifies a hyperplane classifier $\boldsymbol{\omega} \in \mathbb{R}^d$, i.e., a vector $\boldsymbol{\omega}$ that separates all $\boldsymbol{x} \in \mathbb{R}^d$ into two classes A_+ and A_- via the relation

$$\operatorname{sign}(\boldsymbol{\omega}^\top \boldsymbol{x} - \gamma) = \begin{cases} -1 \Rightarrow \boldsymbol{x} \in A_-, \\ +1 \Rightarrow \boldsymbol{x} \in A_+, \end{cases}$$

for a given $\gamma \in \mathbb{R}$. In the half-plane that contains all the good designs, multiple sets I_k, $k = 1, 2, \ldots, n$, are sought with

$$I_k := \left\{ \boldsymbol{x} \in \mathbb{R}^d \,\middle|\, \boldsymbol{\omega}^\top \boldsymbol{x} < \gamma,\, \ell_{k,i} < x_i < u_{k,i},\, i = 1, \ldots, d \right\}.$$

Each set I_k constitutes a *rule* for good designs. The advantage of these rules over the classifier found by the SVM is that they are more intuitive and easier to understand for humans. The authors of [31] state two criteria for the rules I_k that maximize either the volume or the number of training points covered by them. By applying the technique of Lagrange multipliers to these criteria, the authors are able to construct an algorithm that iteratively generates a matching set

$$\bigcup_{k=1}^{n} I_k$$

of n rules.

Cluster Analysis and Permissible Hypercuboids

The method proposed in [36] requires a set of designs $\mathcal{X} := \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subset \mathbb{R}^d$ and applies cluster analysis (see [6, 48]) to them. For this method, the designs that fulfill all given constraints are collected in a set $\mathcal{H}^+$. The authors call these designs *permissible*. Designs which violate at least one of the given constraints are collected in a set of *non-permissible designs* $\mathcal{H}^-$ (see Figure 1.4, top left, permissible designs

are green, non-permissible designs are red). Then, the set $\mathcal{H}^+$ is divided into a given number K of *clusters* C_i such that $\mathcal{H}^+ = \bigcup_{i=1}^{K} C_i$ with $C_i \cap C_j = \emptyset$ for $i \neq j$ and $C_i \neq \emptyset$ for all $i, j = 1, \ldots, K$. For example, in Figure 1.4, top right, the permissible designs are divided into $K = 3$ clusters, marked as squares, diamonds and triangles. A cluster C_i comprises all designs which are most similar to each other and most different from the designs of other clusters.

The clusters are computed by the *K-means clustering algorithm*, see [6, 48, 63]. It finds a set $\mathcal{C} := \{C_1, \ldots, C_K\}$ containing K clusters that partition $\mathcal{X}$. The algorithm does so by finding a partition $\mathcal{C}$ where the squared distance J between the points in a cluster C_i and their mean $\boldsymbol{\mu}_i$ is minimized, where

$$J(C_i) := \sum_{\boldsymbol{x} \in C_i} \|\boldsymbol{x} - \boldsymbol{\mu}_i\|^2.$$

The K-means clustering algorithm then solves the optimization problem

$$\sum_{C_i \in \mathcal{C}} J(C_i) \to \min_{\mathcal{C}}.$$

Each cluster $C_i \in \mathcal{C}$ is a collection of permissible designs and is used to identify regions of permissible designs. The most useful type of region for a design process is a hypercuboid. A simple hypercuboid that can be constructed for each cluster C_i is a bounding box,

$$B_i := \left\{ \boldsymbol{y} \in \mathbb{R}^d \;\middle|\; \min_{\boldsymbol{x} \in C_i} x_j \leq y_j \leq \max_{\boldsymbol{x} \in C_i} x_j,\ j = 1, \ldots, d \right\}.$$

However, the union $\bigcup_{i=1}^{K} B_i$ of hypercuboids is in general not a hypercuboid and may contain a lot of non-permissible design space (Figure 1.4, second row, on the left). Instead, the authors present the following interval approach to generate a suitable hypercuboid from the union of hypercuboids. Each bounding box is a product of intervals, $B_i = I_{i,1} \times \ldots \times I_{i,d}$. Now, every interval is divided into a given number of p subintervals, $I_{i,j} = I_{i,j}^{(1)} \times \ldots \times I_{i,j}^{(p)}$. Thus, each bounding box is a union of p^d subhypercuboids (Figure 1.4, second row, on the right). If we apply multi-index notation to describe this union, we get

$$B_i = \bigcup_{\boldsymbol{k}=1}^{p} H_{i,\boldsymbol{k}},$$

where $H_{i,\boldsymbol{k}} := I_{i,1}^{(k_1)} \times \ldots \times I_{i,d}^{(k_d)}$ and $I_{i,j}^{(k_n)} := \left[a_{i,j}^{(k_n)}, b_{i,j}^{(k_n)} \right]$. In the next step, each subhypercube $H_{i,\boldsymbol{k}}$ is paired with all other subhypercubes $H_{j,\boldsymbol{\ell}}$ and the bounding box $H_{i,j,\boldsymbol{k},\boldsymbol{\ell}}$ of those subhypercubes is calculated,

$$\begin{aligned} H_{i,j,\boldsymbol{k},\boldsymbol{\ell}} := &\left[\min\left\{ a_{i,1}^{(k_1)}, a_{j,1}^{(\ell_1)} \right\}, \max\left\{ b_{i,1}^{(k_1)}, b_{j,1}^{(\ell_1)} \right\} \right] \times \ldots \\ &\times \left[\min\left\{ a_{i,d}^{(k_d)}, a_{j,d}^{(\ell_d)} \right\}, \max\left\{ b_{i,d}^{(k_d)}, b_{j,d}^{(\ell_d)} \right\} \right]. \end{aligned}$$

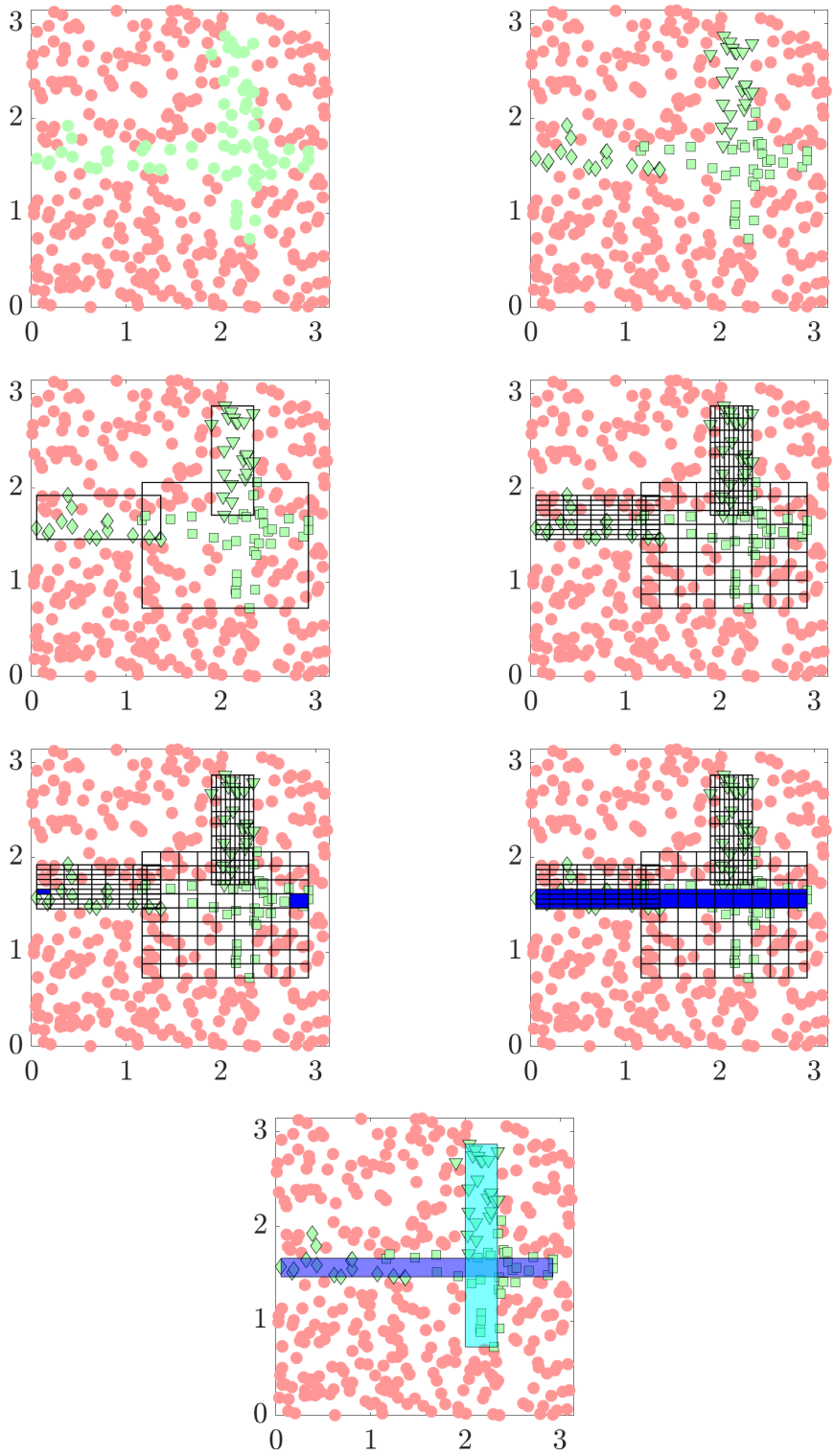

Figure 1.4: A sketch that illustrates the principles of cluster analysis and permissible hypercuboids.

Figure 1.4, third row, shows two chosen subhypercuboids in blue and the bounding box constructed around them. It is also checked whether a bounding box is permissible, i.e., we require that $H_{i,j,\boldsymbol{k},\boldsymbol{\ell}} \cap \mathcal{H}^- = \emptyset$. For each cluster C_i, the permissible bounding box $H_{i,j,\boldsymbol{k},\boldsymbol{\ell}}$ with either the largest volume or the largest minimal interval width is chosen as hypercuboid. This results in at most K hypercuboids which identify regions of permissible designs that can be applied in the design process. In Figure 1.4, bottom, the cluster of squares and triangles yield the same hypercuboid (in light blue), while the cluster of diamonds yields a different hypercuboid.

Space-Based Design Methodology

A methodology for set-based concurrent engineering is presented in [58]. With its help, designers can define how strongly they prefer some designs over others, and the performance of the resulting designs can be measured.

First, for each variable x_i of a design, $i = 1, \dots, n$, a *design parameter* $X_i := [a_i, b_i] \subset \mathbb{R}$, is identified. It represents the range of possible values that can be attained by the variable x_i. Then, each design parameter X_i is associated with a *preference number* $\mathcal{X}_i$, defined as

$$\mathcal{X}_i := \big\{(x_i, p_{\mathcal{X}_i}(x_i)) \,\big|\, x_i \in X_i,\, p_{\mathcal{X}_i}(x_i) : X_i \to [0,1]\big\}.$$

Here, $p_{\mathcal{X}_i}$ is the *preference function*. It allows designers to express their opinion on how desirable certain values x_i admitted by the parameter X_i are for the final design. The preference number is similar to a probability density function from stochastics or a membership function from fuzzy set theory. There, membership functions determine the degree of the membership of an element in a set by mapping it to $[0, 1]$ (see [62]).

In a second step, each preference function $p_{\mathcal{X}_i}(x)$ is divided into $m + 1$ *preference levels* p_j, $j = 0, \dots, m$, with $p_0 := 0$, $p_j := p_{j-1} + \Delta p$ and $\Delta p := \max\limits_{x_i \in X_i} p_{\mathcal{X}_i}(x_i)/m$. Then, the so-called *interval propagation theorem* (IPT, see [28]) is applied to find the *decomposed preference number* $\mathbf{X}_i := \left\{\mathcal{X}_i^{(0)}, \dots, \mathcal{X}_i^{(m)}\right\}$ with

$$\mathcal{X}_i^{(j)} := \big\{x_i \in X_i \,\big|\, p_{\mathcal{X}_i}(x_i) \geq p_j\big\}.$$

Clearly, these sets define regions of the design space that are preferred by the designers up to a certain preference level p_j. Nevertheless, these regions may still contain a lot of unfeasible designs which should be avoided in the design process. Thus, each preference number $\mathcal{X}_i$ is also associated with a so-called *possibilistic distribution* $q_{\mathcal{X}_i}(x_i)$ that measures its performance.

The authors then continue by presenting a variety of measures that weigh the preference and performance of the elements $\mathcal{X}_i^{(j)}$ of a decomposed preference number $\mathbf{X}_i$, such as the *design preference index* (DPI, see also [15]), defined as

$$DPI\left(\mathcal{X}_i^{(j)}\right) := \int_{p_{\mathcal{X}_i}(x_i) \geq p_j} p_{\mathcal{X}_i}(x_i) q_{\mathcal{X}_i}(x_i)\, \mathrm{d}x_i.$$

Approach with Fuzzy Arithmetic and Cluster Analysis

The approach presented in [2] applies fuzzy arithmetic to quantify the uncertainty regarding the structural robustness of a given set of designs. Then, cluster analysis is applied to divide the designs into permissible and non-permissible designs. Inside these clusters, hypercubes containing only permissible designs are constructed. The authors continue by formulating three criteria for an optimal design: First, it should fulfill traditional optimization objectives like minimum cost and maximum aesthetics "in the mean". Second, it should be as robust as possible. Third, it should provide the designers with preferably large decision margins, i.e., there should be as many nearby permissible designs as possible. Finally, a combination of the three criteria allows the identification of an optimal design from within one of the hypercubes.

1.4 Solution Spaces

The methods described previously have certain restrictions. In order to apply the method from [31], the data must be linearly separable. The method presented in [36] becomes very expensive in high dimensions due to the curse of dimensionality. Finally, the procedures described in [2] and [58] require additional information in the form of membership functions and preference functions, respectively, which may not be available.

In this section, we present thus an approach from set-based design that has none of these restrictions. It is the focus of this thesis and lays the foundation for the algorithms presented in the following chapters. This method optimizes high-dimensional subsets of feasible designs, so-called *solution spaces*, see [85]. Synonyms for solution spaces are *permissible design spaces*, *feasible design areas* or *feasible solution sets*, cf. [21, 39, 65].

Definition 1.4.1. *A set $\Omega \subset \Omega_{\text{ds}}$ is called a* solution space *if the objective function f admits only subcritical output values for all designs in Ω, i.e., if*

$$f(\boldsymbol{x}) \leq c$$

for all $\boldsymbol{x} \in \Omega$ and a given value $c \in \mathbb{R}$.

Similar to the methods in Section 1.3, solution spaces try to maximize the size of a set containing admissible designs. Methods that apply solution spaces require no information about the uncertainty of a problem, e.g. the knowledge of a probability distribution function. Thus, they can be applied in the early stages of a design process, where epistemic uncertainty is prevalent. Aleatoric uncertainty, which can be found in later stages of a design process, can also be handled with solution spaces, see [86].

An iterative method to find solution spaces is proposed in [85] and studied extensively in [40, 42]. Another method that can calculate solution spaces directly if f is at most a quadratic function with linear constraints is given in [26, 27]. Finally, a method where the design space is decomposed into 2D-subpsaces is considered in [24, 25].

We now give a few basic definitions and the fundamental problem statement to work with the method described in [85].

Definition 1.4.2. *A design* $\boldsymbol{x} \in \Omega_{\text{ds}}$ *is called a* good design *or a* good design point *if* $f(\boldsymbol{x}) \leq c$ *and a* bad design *or a* bad design point *if* $f(\boldsymbol{x}) > c$ *for a critical value* $c \in \mathbb{R}$ *which is given by the problem. Additionally, the set of all good designs is defined as the* complete solution space

$$\Omega_c := \big\{\boldsymbol{x} \in \Omega_{\text{ds}} : f(\boldsymbol{x}) \leq c\big\}.$$

The key idea of [85] is to express the solution space sought by intervals for each input design variable, thus representing a high-dimensional, axis-parallel box Ω_{box}:

$$\Omega_{\text{box}} := \prod_{i=1}^{d} [a_i, b_i] \subset \Omega_{\text{ds}}.$$

Therefore, a design will always be classified to be good, as long as the values of its design variables remain within their respective intervals. By specifying target intervals that do not depend on the choice of interacting design variables, design variables are said to be uncoupled, enabling the independent development of the involved components. As the size of the intervals and thus the volume of the box are to be maximized, the following semi-infinite optimization problem can be formulated:

$$\begin{cases} \text{Maximize the volume } \mu(\Omega_{\text{box}}) \\ \text{over all axis-parallel boxes } \Omega_{\text{box}} \subset \Omega_{\text{ds}} \\ \text{subject to } f(\boldsymbol{x}) \leq c \text{ for all } \boldsymbol{x} \in \Omega_{\text{box}}. \end{cases} \tag{1.4.1}$$

Here, the measure μ is defined as

$$\mu(\Omega_{\text{box}}) := \prod_{i=1}^{d} b_i - a_i.$$

The optimal solution space gained from solving the above problem is then used in a development process to identify valid designs. Methods that help to find solution spaces are studied in this thesis and outlined in the following.

1.5 Outline of the Thesis

In Chapter 2, we describe the underlying optimization problem and give a detailed account of the box optimization algorithm which was developed to solve that problem. We demonstrate its functionality on two experiments and carry out a parameter study to find a good choice for the growth rate.

In Chapter 3, we introduce 2D-maps and develop the rotated box optimization algorithm. We show how to apply principal component analysis to rotate boxes on 2D-maps and explain the differences to the box optimization algorithm. We apply the rotated box optimization algorithm to the same two problems as the box optimization algorithm. The experiment in Subsection 3.4.2 is carried out to confirm that, in the mean, the algorithm finds the correct angle for a rotated box inside the good design space.

In Chapter 4, we present the polytope optimization algorithm. We explain in detail several techniques to manipulate 2D polygons. These techniques are necessary to formulate the algorithm. It is conceptually similar to the rotated box optimization algorithm. However, it requires a large number of adjustments because the rotated boxes are replaced by polytopes. We use the polytope optimization algorithm to solve the same two example problems as for the box optimization algorithm and the rotated box optimization algorithm. Because the polytope optimization algorithm introduces many new parameters, we conduct a large parameter study that yields good choices for those parameters.

In Chapter 5, we test two modifications that are theoretically applicable to all three algorithms. In one modification, we swap the order of iteration in the algorithms. For the other modification, we analyze the covariance of a few design points before we initialize the optimization algorithm. Here, the idea is that we find a reasonable coupling for 2D-maps. This coupling can be used in the case when none is given. It is also possible to find one that is even better than a given coupling.

In Chapter 6, we compare the box optimization algorithm, the rotated box optimization algorithm and the polytope optimization algorithm by applying them to three high-dimensional problems. Additionally, we conduct a final parameter study. There, we study the performance of the optimization algorithms when we change the number of sampled designs and the number of steps in the exploration and the consolidation phase.

We give a conclusion to the thesis and a few final remarks in Chapter 7. Additionally, we give an outlook on possible future work.

Chapter 2

Box Optimization

The algorithm presented in this chapter is introduced as "A Search Algorithm for Solution Spaces" in [85] and is the object of further study in [23, 40, 41, 42]. Since this thesis extends the algorithm and is hence based on it, we call it the *box optimization algorithm*.

The idea of the box optimization algorithm is the following: because we want to solve problem (1.4.1), we initialize a hyperbox Ω_{box} somewhere in the design space Ω_{ds}. Next, we want to move this box through Ω_{ds} until we have found a spot with a large amount of good design space. In addition, we may want to adjust the size of the box, as this good design space may be bigger or smaller than the initial box. The process of moving the box Ω_{box} through Ω_{ds} and adjusting its size is done in the *exploration phase*. To this end, we sample a fixed number of design points in Ω_{box} and then adjust its size by *trimming* it such that it only contains good sampled design points. Immediately afterwards we let it *grow* again, thereby trying to find new good design space. By repeating the sampling, trimming and growing steps, the box moves through and explores the design space. We stop this repetition after a certain number of steps which ends the exploration phase.

After the exploration phase, the box may still contain bad design space. We consolidate it as an acceptable solution to (1.4.1) by repeating the sampling and trimming steps several times. This is the *consolidation phase*. The box does not move in this phase because it is not grown anymore. The design space inside the final box obtained from the consolidation phase will be mostly good and the probability of sampling a bad design will be very low, which we are going to show in Section 2.3. The flowchart in Figure 2.1 summarizes the steps of the box optimization algorithm.

In the following, we explain the box optimization algorithm. Additionally, we show a few theoretical results and present some numerical experiments.

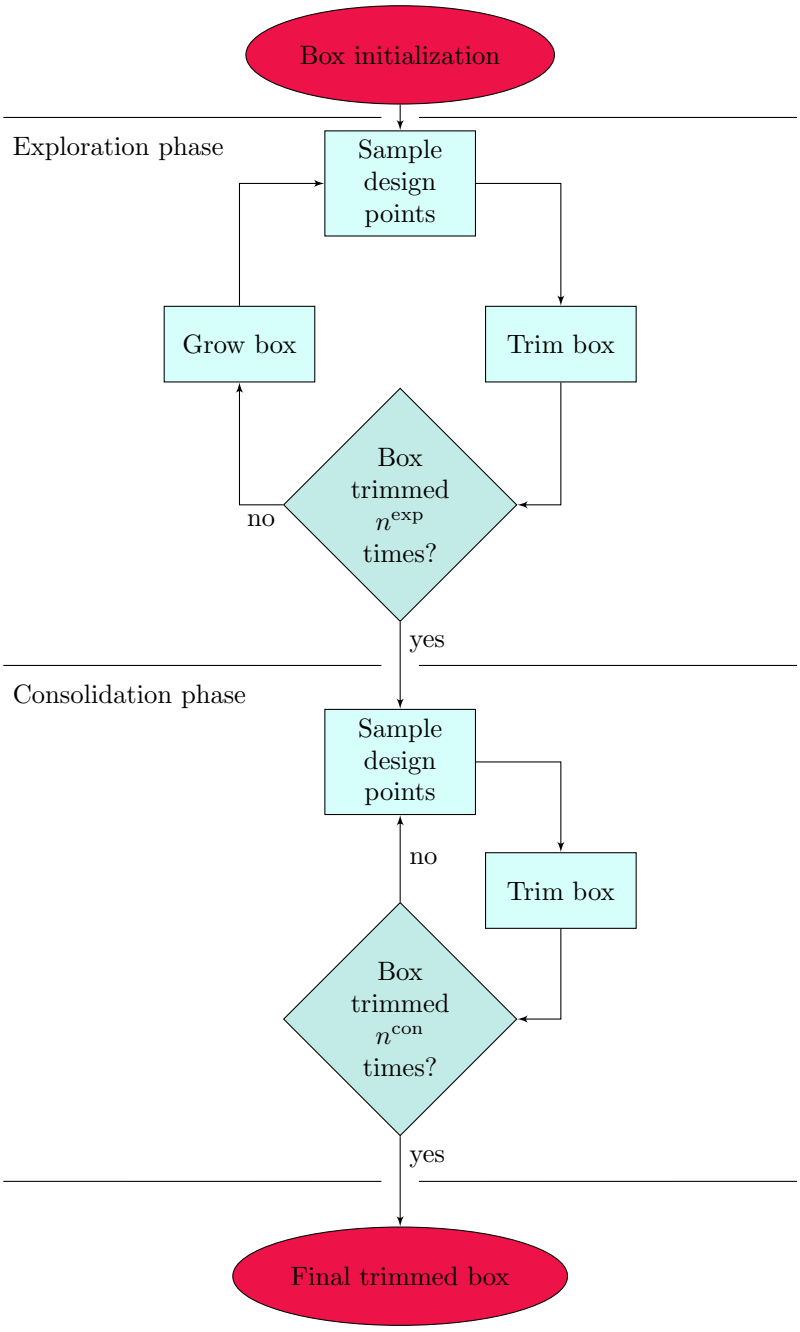

Figure 2.1: Flowchart of the box optimization algorithm.

2.1 Algorithm

2.1.1 Box Initialization

In the beginning of the algorithm, an initial hyperbox is required. It is either given by the problem or can be constructed by other means. For example, a classical optimization algorithm or a genetic algorithm, such like differential evolution, compare [42, 73], can be used to find an initial optimal design point. Then, a hyperbox can be constructed with the optimal design point as its center. In [42], it is recommended that the initial box is not too large, otherwise it may be possible that no good design points are sampled inside in which case the algorithm immediately terminates. For example, if the optimum lies in a small, isolated region of good designs, the box should not be chosen much larger than that region. If a box is found, it is used as the initial box for the iteration in the next phase.

2.1.2 Exploration Phase

In the exploration phase, designs $\boldsymbol{x}$ are sampled within Ω_{box} and subsequently evaluated by applying f. This divides them into good and bad design points according to Definition 1.4.2. Then, Ω_{box} is trimmed such that the bad design points are removed. Finally, the box is grown again. By repeating these steps n^{exp} times, the hyperbox moves through the design space Ω_{ds} and finds a large region of good design space.

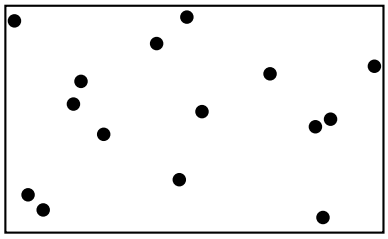

Figure 2.2: A few designs sampled inside a box.

Sample Design Points

Inside Ω_{box}, N uniformly distributed random design points are sampled, compare Figure 2.2. Afterwards, the design points are evaluated. All of the design points $\boldsymbol{x}$ with an objective value $f(\boldsymbol{x}) \leq c$ are collected in the set of good designs

$$\mathcal{X}^{\text{good}} := \left\{ {}_{(1)}\boldsymbol{x}^{\text{good}}, \ldots, {}_{(n^{\text{good}})}\boldsymbol{x}^{\text{good}} \right\},$$

and all of the design points $\boldsymbol{x}$ with an objective value $f(\boldsymbol{x}) > c$ are collected in the set of bad designs

$$\mathcal{X}^{\text{bad}} := \left\{ {}_{(1)}\boldsymbol{x}^{\text{bad}}, \ldots, {}_{(n^{\text{bad}})}\boldsymbol{x}^{\text{bad}} \right\},$$

see Figure 2.3. The sets are sorted with respect to the objective values of the sample design points, in descending order.

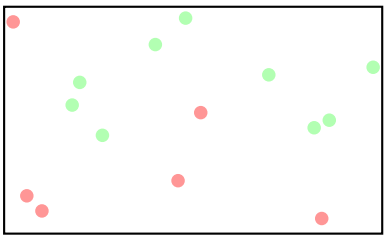

Figure 2.3: Good designs are colored in green, bad designs in red.

Trim Box

Next, Ω_{box} is trimmed by moving its boundaries onto the bad design points until there are only good design points left. Because there is no unique way to do this, multiple boxes $\Omega^\star_{\text{box}}$ are calculated, such that for each good design point $\boldsymbol{x}^{\text{good}} \in \mathcal{X}^{\text{good}}$ there exists at least one trimmed box $\Omega^\star_{\boldsymbol{x}^{\text{good}}}$ that contains it, cf. [40]. From those boxes, the one with the largest volume is chosen as the final box $\Omega^\star_{\text{box}}$.

The *box trimming algorithm* (Algorithm 1) gives a detailed account of how the box is trimmed. Its steps are explained below.

Algorithm 1 (Box Trimming). This algorithm trims the box such that the smallest number of good design points is removed.

1: **Input:** Ω_{box}, $\mathcal{X}^{\text{good}}$, $\mathcal{X}^{\text{bad}}$
2: **Output:** $\Omega^\star_{\text{box}}$

3: **for all** $\boldsymbol{x}^{\text{good}} \in \mathcal{X}^{\text{good}}$ **do**
4: $\quad\Omega^\star_{\boldsymbol{x}^{\text{good}}} \leftarrow \Omega_{\text{box}}$
5: $\quad\prod_{i=1}^{d} [a_i^\star, b_i^\star] \leftarrow \Omega^\star_{\boldsymbol{x}^{\text{good}}}$
6: $\quad$**for all** $\boldsymbol{x}^{\text{bad}} \in \mathcal{X}^{\text{bad}}$ **do**
7: $\quad\quad[\boldsymbol{n}^{\text{good}}, \boldsymbol{n}^{\text{bad}}] \leftarrow \texttt{countpoints}\big(\boldsymbol{x}^{\text{good}}, \boldsymbol{x}^{\text{bad}}, \Omega^\star_{\boldsymbol{x}^{\text{good}}}, \mathcal{X}^{\text{good}}, \mathcal{X}^{\text{bad}}\big)$
8: $\quad\quad I_{\text{good}} \leftarrow \left\{ i \in \{1, \dots, d\} \;\middle|\; n_i^{\text{good}} = \min_{j \in \{1,\dots,d\}} n_j^{\text{good}} \right\}$
9: $\quad\quad I_{\text{bad}} \leftarrow \left\{ i \in I_{\text{good}} \;\middle|\; n_i^{\text{bad}} = \max_{j \in I_{\text{good}}} n_j^{\text{bad}} \right\}$
10: $\quad\quad i^\star \in_{\text{rand}} I_{\text{bad}}$
11: $\quad\quad$**if** $x_{i^\star}^{\text{bad}} < x_{i^\star}^{\text{good}}$ **then** $a_{i^\star}^\star \leftarrow x_{i^\star}^{\text{bad}}$ **else** $b_{i^\star}^\star \leftarrow x_{i^\star}^{\text{bad}}$ **end if**
12: $\quad$**end for**
13: $\quad\mathcal{X}_{\cap}^{\text{good}} \leftarrow \mathcal{X}^{\text{good}} \cap \prod_{i=1}^{d} [a_i^\star, b_i^\star]$
14: $\quad$**for all** $a_i^\star \neq a_i$ **do** $a_i^\star \leftarrow \min_{\boldsymbol{x}^{\text{good}} \in \mathcal{X}_{\cap}^{\text{good}}} x_i^{\text{good}}$ **end for**
15: $\quad$**for all** $b_i^\star \neq b_i$ **do** $b_i^\star \leftarrow \max_{\boldsymbol{x}^{\text{good}} \in \mathcal{X}_{\cap}^{\text{good}}} x_i^{\text{good}}$ **end for**
16: **end for**
17: $\Omega^\star_{\text{box}} \leftarrow \underset{\Omega^\star_{\boldsymbol{x}^{\text{good}}}}{\arg\max}\, \mu\big(\Omega^\star_{\boldsymbol{x}^{\text{good}}}\big)$

The algorithm requires an axis-parallel hyperbox $\Omega_{\text{box}} = \prod_{i=1}^{d} [a_i, b_i]$ as well as sets of good and bad design points $\mathcal{X}^{\text{good}}$ and $\mathcal{X}^{\text{bad}}$ as inputs (line 1). The output (line 2) is a trimmed box $\Omega^\star_{\text{box}} := \prod_{i=1}^{d} [a_i^\star, b_i^\star]$. The algorithm iterates over all good design points $\boldsymbol{x}^{\text{good}}$ (line 3), initializes a new box $\Omega^\star_{\boldsymbol{x}^{\text{good}}}$ in lines 4 and 5, and then begins to loop over the bad design points $\boldsymbol{x}^{\text{bad}}$ in line 6. For each bad design point $\boldsymbol{x}^{\text{bad}}$ and

each dimension i, it counts how many design points would get removed if the box is trimmed to x_i^{bad} in line 7. Thereby, we use the procedure `countpoints`, which is introduced in Algorithm 2, the *count points algorithm*, and described below. Then, it finds the dimensions where the fewest good design points are removed (line 8), chooses from those the dimensions where the most bad design points are removed (line 9), and finally, if it has not found a unique dimension yet, chooses one of those dimensions at random (line 10). In line 11, the box is trimmed in the chosen dimension to the current bad design point such that the current good design point does not get removed. For a visualization of this process, we refer to Figure 2.4.

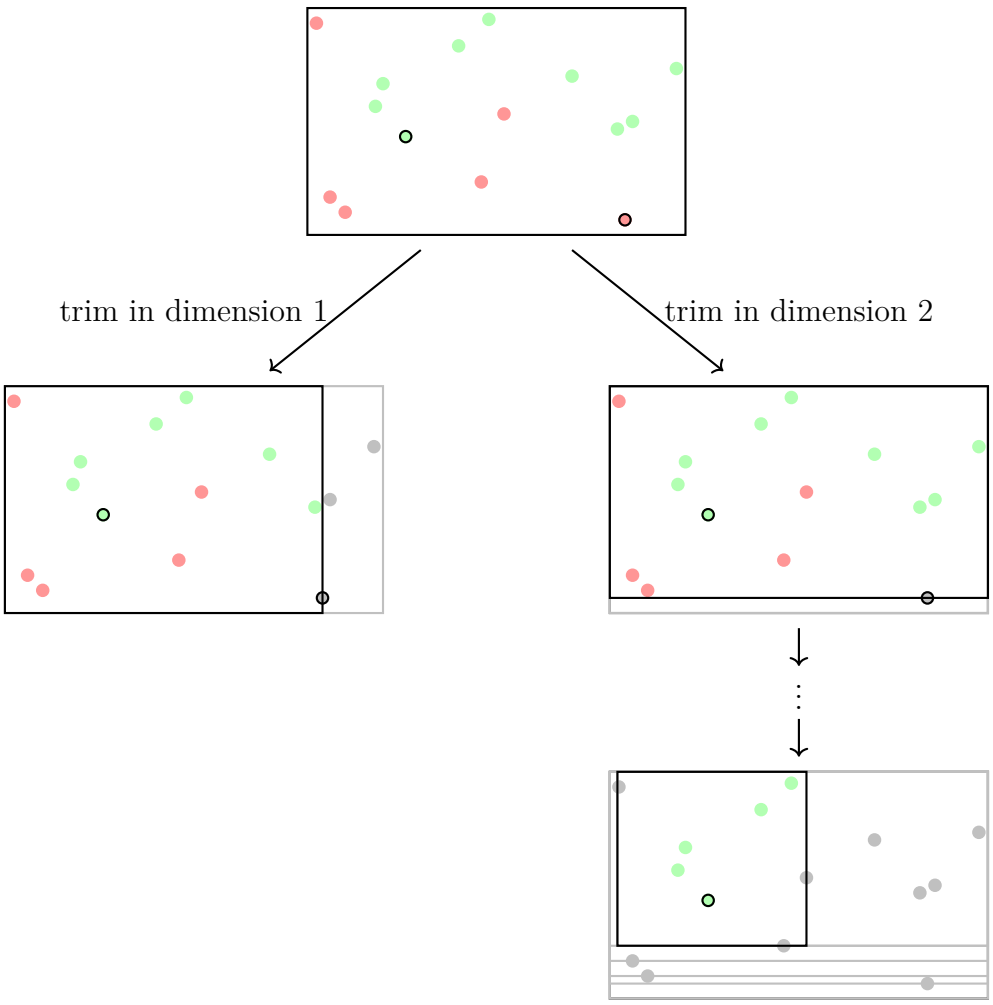

Figure 2.4: For two chosen good and bad design points (marked by black circles), the box is trimmed in dimensions 1 (left) and 2 (right). The right box contains more good designs, so the algorithm proceeds to trim more bad designs from it. The final result is the box at the bottom.

After having iterated over all bad design points, the remaining good design points are collected (line 13) and the boundaries are trimmed further to the nearest good design points in dimensions where the boundaries actually had to be trimmed (lines 14 and 15, compare Figure 2.5).

Finally, after iterating over all good design points, the box $\Omega^{\star}_{\boldsymbol{x}^{\text{good}}}$ with the largest volume μ is chosen as the output (line 17, see also Figure 2.6).

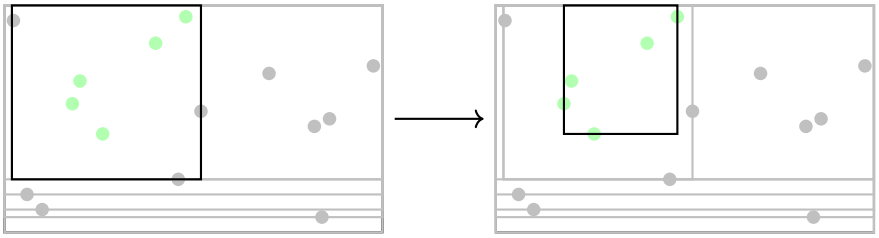

Figure 2.5: The box is trimmed in every dimension from every direction it has previously been trimmed.

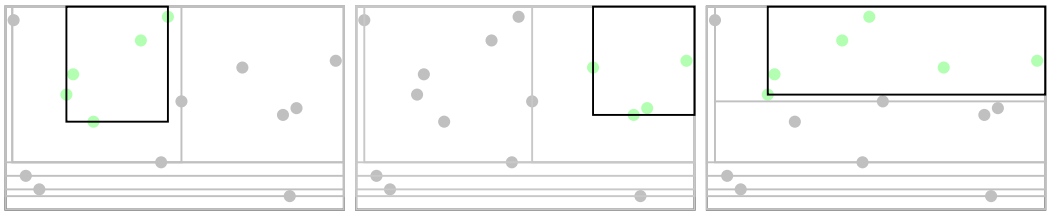

Figure 2.6: For the example from Figure 2.4, the algorithm yields the three boxes shown above. The rightmost box has the largest volume, so it is chosen as the output of the trimming algorithm.

The procedure `countpoints`, implemented by Algorithm 2, counts the removed design points if the box is trimmed to x_i^{bad} in dimension i. It takes a good design point $\boldsymbol{x}^{\text{good}}$, a bad design point $\boldsymbol{x}^{\text{bad}}$, a solution space Ω_{box}, and two sets of good and bad design points as inputs. The outputs are two vectors $\boldsymbol{n}^{\text{good}}$ and $\boldsymbol{n}^{\text{bad}}$ that count how many good and bad design points get removed if the boundary of Ω_{box} in dimension i is moved onto x_i^{bad}, while the design point $\boldsymbol{x}^{\text{good}}$ is left inside Ω_{box}. To this end, the algorithm iterates over all dimensions (see line 3), and decides whether $x_i^{\text{bad}} < x_i^{\text{good}}$ or $x_i^{\text{bad}} \geq x_i^{\text{good}}$ (lines 4 and 7). Then, it counts the number of design points in dimension i which lie between the boundary and x_i^{bad} (lines 5–9).

Algorithm 2 (Count Points). This algorithm counts the design points which are removed by the trimming step.

1: **Input:** $\boldsymbol{x}^{\text{good}}, \boldsymbol{x}^{\text{bad}}, \Omega_{\text{box}}, \mathcal{X}^{\text{good}}, \mathcal{X}^{\text{bad}}$
2: **Output:** $\boldsymbol{n}^{\text{good}}, \boldsymbol{n}^{\text{bad}}$

3: **for** $i = 1, \ldots, d$ **do**
4: **if** $x_i^{\text{bad}} < x_i^{\text{good}}$ **then**
5: $n_i^{\text{good}} \leftarrow \#\left\{ \boldsymbol{x} \in \mathcal{X}^{\text{good}} \;\middle|\; a_i \leq x_i \leq x_i^{\text{bad}} \right\}$
6: $n_i^{\text{bad}} \leftarrow \#\left\{ \boldsymbol{x} \in \mathcal{X}^{\text{bad}} \;\middle|\; a_i \leq x_i \leq x_i^{\text{bad}} \right\}$
7: **else**
8: $n_i^{\text{good}} \leftarrow \#\left\{ \boldsymbol{x} \in \mathcal{X}^{\text{good}} \;\middle|\; x_i^{\text{bad}} \leq x_i \leq b_i \right\}$
9: $n_i^{\text{bad}} \leftarrow \#\left\{ \boldsymbol{x} \in \mathcal{X}^{\text{bad}} \;\middle|\; x_i^{\text{bad}} \leq x_i \leq b_i \right\}$
10: **end if**
11: **end for**

Grow Box

The last part of one exploration step k is growing the box. Each interval $[a_i, b_i]$ is stretched by a growth rate $g^{(k)}$ such that

$$\Omega_{\text{box}} := \prod_{i=1}^{d} \left[\overline{a}_i, \overline{b}_i\right]$$

with

$$\overline{a}_i := a_i - g^{(k)} \cdot (b_i - a_i), \quad \overline{b}_i := b_i + g^{(k)} \cdot (b_i - a_i),$$

see also Figure 2.7.

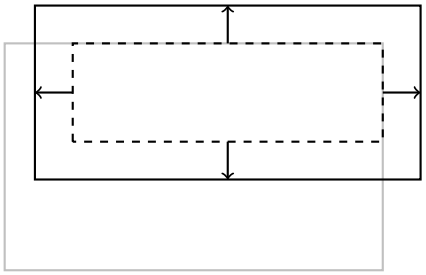

Figure 2.7: The box chosen in Figure 2.6 is grown in all directions.

The growth rate may either be a constant $g^{(0)}$, set at the beginning of the algorithm such that

$$g^{(k)} := g^{(k-1)},$$

or, before growing the box, it may be updated according to the formula

$$g^{(k)} := \frac{a_k^{\text{good}}}{a^{\text{target}}} \cdot g^{(k-1)},$$

where $a_k^{\text{good}} := n_k^{\text{good}}/N$ is the fraction of good design points before trimming the box in exploration step k and a^{target} is the desired fraction of good design points. Subsection 2.3.1 gives more details on how the growth rate should be chosen.

Note that the box will usually not grow arbitrarily large during the exploration phase, as it will always be trimmed before growing. This cycle of trimming and growing essentially makes the box move through the design space Ω_{ds}.

After this growth step, the algorithm returns to the step “Sample Design Points” unless all steps have been iterated n^{exp} times.

2.1.3 Consolidation Phase

The box is no longer grown in this phase, since it is expected to be in a position with a large amount of good design space. However, the box might still contain some bad design space, and the goal of this phase is to remove as much bad design space as necessary. Thus, one step of the consolidation phase consists of sampling design points and then trimming the box. The consolidation phase is terminated

after either a fixed number of n^{con} steps or when no bad design points have been sampled three times in series. Terminating after three of those steps is done to save time, the reasoning being that more consolidation steps do not change the quality of the box by much. The final box from the consolidation phase is then the solution space found by the box opimization algorithm.

2.2 Modifications to the Algorithm

We had to modify the algorithm described in the previous section to keep it compatible with the rotated box optimization algorithm and the polytope optimization algorithm which will be presented in Chapters 3 and 4. However, these modifications do not change the functionality of the algorithm and have no immediate effect on its results. Instead, they make the results more easily comparable to those of the other two algorithms.

- **Mapping to the Unit Cube.** The first modification is done after the design points are sampled and evaluated. The actual box Ω_{box} and the design points are mapped from the design space $\Omega_{\text{ds}} = \prod_{i=1}^{d}[\alpha_i, \beta_i]$ to the unit cube $[0,1]^d$ via the linear mapping

$$\boldsymbol{x} \mapsto \begin{bmatrix} 1/(\beta_1 - \alpha_1) & & & \\ & 1/(\beta_2 - \alpha_2) & & \\ & & \ddots & \\ & & & 1/(\beta_d - \alpha_d) \end{bmatrix} (\boldsymbol{x} - \boldsymbol{\alpha}).$$

  This is done to normalize all design variables. Each interval $[\alpha_i, \beta_i]$ is thereby transformed into an interval $\left[\widetilde{\alpha}_i, \widetilde{\beta}_i\right]$ with $\widetilde{\beta}_i - \widetilde{\alpha}_i = 1$. Thus, all intervals have the same length. After the growth step, the design points and the unit cube $[0,1]^d$ are remapped to Ω_{ds} via the inverse mapping

$$\boldsymbol{x} \mapsto \begin{bmatrix} \beta_1 - \alpha_1 & & & \\ & \beta_2 - \alpha_2 & & \\ & & \ddots & \\ & & & \beta_d - \alpha_d \end{bmatrix} \boldsymbol{x} + \boldsymbol{\alpha}.$$

- **Sampling Inside the Design Space.** The second modification handles the case when the box grows into the exterior of Ω_{ds}, including design variables that are out of scope. In the original box optimization algorithm, this has been solved by simply retracting the axis-parallel box onto the boundary of Ω_{ds}. The box does not lose any good design space in the process. However, in order to keep the box optimization algorithm compatible with the rotated box optimization algorithm, design points are only sampled in $\Omega_{\text{box}} \cap \Omega_{\text{ds}}$ instead of only Ω_{box} in each step of "Sample Design Points". Consequently, we also have to modify how we measure the volume of Ω_{box}. Thus, whenever we have to calculate $\mu(\Omega_{\text{box}})$ in the algorithm, we instead calculate

$$\mu(\Omega_{\text{box}} \cap \Omega_{\text{ds}}) = \prod_{i=1}^{d} \big(\min\{b_i, \beta_i\} - \max\{a_i, \alpha_i\}\big).$$

2.3 Probability of Finding Good Designs

As the algorithm will fail if no good designs are found, we are interested in the probability of this event. Another important quantity is the amount of good design space that is present in the final box from the consolidation phase and how likely it is to draw a good design from that box. This section is concerned with the analysis of these probabilities.

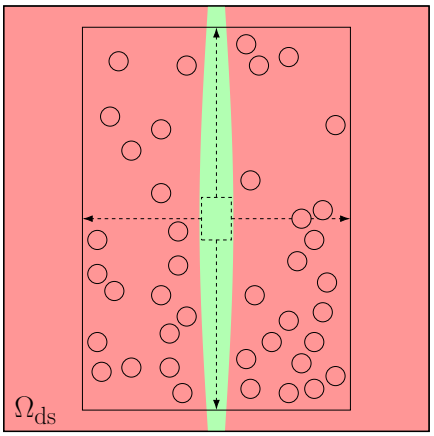

Figure 2.8: Setting the growth rate too large may lead to zero good designs being sampled in the subsequent sampling step.

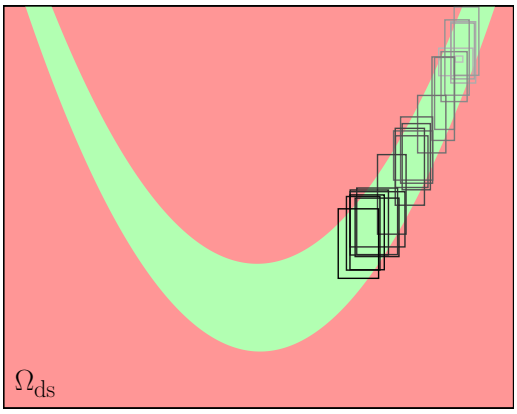

Figure 2.9: When the growth rate is too small, the box moves only very slowly and does not find the bottom of the U-shape, where the good design space is larger (the first few boxes are gray, the later boxes become darker).

2.3.1 In the Growth Step

The growth step plays a critical role in the exploration phase. It allows the box to move into the design space surrounding it. An important parameter here is the

growth rate. If it is chosen too large, no good design points may be found at all (see Figure 2.8 for an illustration). If it is chosen too small, only a small part of Ω_{ds} might be surveyed (compare Figure 2.9) during the whole exploration phase. Thus, a guideline for choosing the growth rate might be of interest. In [40], the following theorem is proven that determines an upper bound for the constant growth rate $g^{(k)}$. Note that we changed the notation from there to match ours.

Theorem 2.3.1. *Let $\Omega_{\text{trim}}^{(k)} := \prod_{i=1}^{d}[a_i^\star, b_i^\star]$ be the output box gained from the trimming step in the k-th iteration of the exploration phase and a_k be the true, usually unknown fraction of good design space inside $\Omega_{\text{trim}}^{(k)}$. Additionally, let $\Omega_{\text{grow}}^{(k)}$ be the result of the subsequent growth step. Define A_{k+1} as the event that the number of good design points n_{k+1}^{good} of N total design points sampled in $\Omega_{\text{grow}}^{(k)}$ in the $(k+1)$-st iteration of the exploration phase is greater than or equal to 1. Then, it holds for the probability of A_{k+1} that*

$$P(A_{k+1}) \geq 1 - \left(1 - \frac{a_k}{(1+2g^{(k)})^d}\right)^N.$$

Proof. Denote by p the (unknown) probability to sample a good design point in $\Omega_{\text{grow}}^{(k)}$. Since we draw the design points from a uniform distribution, we have

$$P(A_{k+1}) = 1 - (1-p)^N.$$

In the worst case, the set $\Omega_{\text{grow}}^{(k)} \setminus \Omega_{\text{trim}}^{(k)}$ contains only bad design space, which yields the inequality

$$p \geq a_k \frac{\mu\left(\Omega_{\text{trim}}^{(k)}\right)}{\mu\left(\Omega_{\text{grow}}^{(k)}\right)}$$

and hence

$$P(A_{k+1}) \geq 1 - \left(1 - a_k \frac{\mu\left(\Omega_{\text{trim}}^{(k)}\right)}{\mu\left(\Omega_{\text{grow}}^{(k)}\right)}\right)^N. \tag{2.3.1}$$

Since the volume of $\Omega_{\text{trim}}^{(k)}$ is

$$\mu\left(\Omega_{\text{trim}}^{(k)}\right) = \mu\left(\prod_{i=1}^{d}[a_i^\star, b_i^\star]\right) = \prod_{i=1}^{d}(b_i^\star - a_i^\star)$$

and the volume of $\Omega_{\text{grow}}^{(k)}$ is

$$\begin{aligned}
\mu\left(\Omega_{\text{grow}}^{(k)}\right) &= \mu\left(\prod_{i=1}^{d}\left[a_i^\star - g^{(k)} \cdot (b_i^\star - a_i^\star), b_i^\star + g^{(k)} \cdot (b_i^\star - a_i^\star)\right]\right) \\
&= \prod_{i=1}^{d}(b_i^\star - a_i^\star)\left(1 + 2g^{(k)}\right) \\
&= \left(1 + 2g^{(k)}\right)^d \prod_{i=1}^{d}(b_i^\star - a_i^\star),
\end{aligned}$$

we obtain

$$\frac{\mu\left(\Omega_{\text{trim}}^{(k)}\right)}{\mu\left(\Omega_{\text{grow}}^{(k)}\right)} = \frac{\prod_{i=1}^{d}(b_i^\star - a_i^\star)}{\left(1+2g^{(k)}\right)^d \prod_{i=1}^{d}(b_i^\star - a_i^\star)} = \frac{1}{\left(1+2g^{(k)}\right)^d}.$$

Inserting this into inequality (2.3.1) yields the assertion. □

From Theorem 2.3.1, [40] concludes that, if $g^{(k)}$ fulfills

$$P(A_{k+1}) \geq 1 - \left(1 - \frac{a_k}{\left(1+2g^{(k)}\right)^d}\right)^N \geq q,$$

the probability of finding a new sample point is at least q. It is also mentioned that a_k is usually unknown and has hence to be estimated by a confidence interval $[a_k^{\text{low}}, a_k^{\text{up}}]$ to the confidence level r. It is concluded that, if at least one good design point should be found with probability q/r, $g^{(k)}$ should be chosen such that

$$0 \leq g^{(k)} \leq \frac{1}{2}\left(\left(\frac{a_k^{\text{low}}}{1-(1-q/r)^{1/N}}\right)^{1/d} - 1\right). \tag{2.3.2}$$

It turns out that applying this inequality to find a suitable growth rate is problematic. The expression on the right side of (2.3.2) tends to 0 as the dimension d grows. This means that either $g^{(k)}$ has to be chosen very small, or the number of sample points N has to be very large. The latter option poses a huge computational cost, as it requires more evaluations of the objective function f. The former option implies that the box will move only slowly through the design space. It is particularly slow in case the growth rate is constant, e.g. $g^{(0)} = g^{(1)} = \ldots = g^{(n^{\text{exp}})} =: g$, since g has to be chosen such that (2.3.2) is valid for each exploration step k.

As an alternative, [40] and [42] recommend a *dynamic* choice of the growth rate:

$$g^{(k)} := \frac{a_k^{\text{good}}}{a^{\text{target}}} \cdot g^{(k-1)}. \tag{2.3.3}$$

It couples the growth rate in the exploration step k to the previous growth rate and the amount of good designs found in the step $k-1$. The numerator $a_k^{\text{good}} := n_k^{\text{good}}/N \in [0,1]$ is the fraction of good design points and is calculated anew after the design points are sampled in the exploration step k. The denominator $a^{\text{target}} \in (0,1]$ is chosen at the beginning of the algorithm and does not change. It denotes the fraction of good design points the algorithm should aim to find in each exploration step and is responsible for the general speed with which the box moves through Ω_{ds}.

The following behavior can be observed for the parameter a^{target} (compare also Subsection 2.4.2): If it is close to 1, the growth rate will, according to our experience, increase only occasionally and the box will move very slowly. Thus, trying to find a box containing only good points, i.e., setting $a^{\text{target}} = 1$, is – seemingly paradoxically – not desirable during the exploration phase. If a^{target} is smaller than 1, the box will move faster and in larger steps. Indeed, the algorithm may even break down if the growth rate becomes too large and no good designs are found (see again Figure 2.8).

The fraction $a_k^{\text{good}}/a^{\text{target}}$ allows us to draw conclusions about the next exploration step. If $a_k^{\text{good}}/a^{\text{target}} \gg 1$, much more good designs than desired were sampled, likely because the box has found a relatively large region of good design space. Increasing the growth rate lets the box probe the boundaries of that region faster, which in turn means that fewer design points have to be sampled overall. On the other hand, if $a_k^{\text{good}}/a^{\text{target}} \ll 1$, far too few good designs have been sampled, indicating that the box has grown too much. Reducing the growth rate lets the box retract into good design space, as it will be trimmed more than it grows.

In general, the dynamic growth rate provides a reasonable growth speed while ensuring that the box stays mostly within good design space. Thus, it should be favored over a constant growth rate.

2.3.2 In the Consolidation Phase

During the consolidation phase, the amount of good design space inside the box is of interest, as the phase can be stopped early if only good design points have been sampled three times in series. This termination condition is derived from the theorem below, originally proven in [53] and formalized in [40]. The proof applies Bayesian inference to derive a credible interval (see [4, 32]).

Definition 2.3.2. *Given a random vector $\boldsymbol{x}$ from a set of possible values $\boldsymbol{X}$, an unknown parameter θ with density $p(\theta)$ and two functions $u, v : \boldsymbol{X} \to \mathbb{R}$, we define the interval $C := \big[u(\boldsymbol{x}), v(\boldsymbol{x})\big]$ as the* credible interval *with* credible level α *if*

$$P\big(u(\boldsymbol{x}) < \theta < v(\boldsymbol{x}) | \boldsymbol{x}\big) := \int_{u(\boldsymbol{x})}^{v(\boldsymbol{x})} p(\theta|\boldsymbol{x}) \,\mathrm{d}\theta = 1 - \alpha.$$

Note that a credible interval differs slightly from a *confidence interval*, which is defined as the interval $D := \big[u(\boldsymbol{x}), v(\boldsymbol{x})\big]$ such that

$$P\big(u(\boldsymbol{x}) < \theta < v(\boldsymbol{x}) | \theta\big) := \int_{u(\boldsymbol{x})}^{v(\boldsymbol{x})} p(\theta|\theta) \,\mathrm{d}\theta = 1 - \alpha$$

for all θ. Thus, a credible interval C treats the parameter θ as a random variable that lies within C with probability $1-\alpha$. In contrast, a confidence interval D treats θ as a fixed value and D encloses θ with probability $1 - \alpha$ (one can be "reasonably confident" that $\theta \in D$). However, whether this is actually the case for a particular θ, is entirely uncertain (see again [4]).

For the following theorem, let a_k denote the unknown true fraction of good design space within the box Ω_{box} before trimming it in the k–th step of the consolidation phase. Before we sample any design points in Ω_{box}, we are unable to make any assertions about the probability of sampling a good design point because we treat f as a black-box function. Thus, the value of a_k is entirely uncertain and we treat it as continuous random variable that admits values between 0 and 1. Since there is no other information available, it is reasonable to model the prior probability $p(a_k)$ to be uniformly distributed. The only information required by the uniform distribution is the range of its values, which in this case is $[0, 1]$. Other bounded probability distributions would require more knowledge about a_k, e.g. the beta distribution requires two shape parameters α and β (see, for example, [34]).

Theorem 2.3.3. *Let n_k^{good} be the amount of good design points among N sampled design points inside Ω_{box} in the k–th consolidation step. Then, the probability that a_k lies within a given interval $[a_{\text{low}}, a_{\text{up}}]$ is*

$$P\Big(a_{\text{low}} < a_k < a_{\text{up}} \Big| n_k^{\text{good}}\Big) = \frac{\displaystyle\int_{a_{\text{low}}}^{a_{\text{up}}} t^{n_k^{\text{good}}} (1-t)^{N-n_k^{\text{good}}} \,\mathrm{d}t}{\displaystyle\int_0^1 s^{n_k^{\text{good}}} (1-s)^{N-n_k^{\text{good}}} \,\mathrm{d}s}.$$

Proof. We set $n_k := n_k^{\text{good}}$ to simplify the notation. The probability of sampling n_k designs from N total designs is modelled with the posterior distribution $p(a_k|n_k)$. With $p(n_k) \neq 0$, it holds

$$p(a_k|n_k) = \frac{p(a_k \wedge n_k)}{p(n_k)}.$$

Applying Bayes' theorem, we get

$$p(a_k|n_k) = \frac{p(n_k|a_k)p(a_k)}{p(n_k)},$$

where $p(a_k)$ is the prior distribution. In view of

$$p(n_k) = \int_0^1 p(n_k|s)p(s) \,\mathrm{d}s,$$

we find

$$p(a_k|n_k) = \frac{p(n_k|a_k)p(a_k)}{\displaystyle\int_0^1 p(n_k|s)p(s) \,\mathrm{d}s}.$$

Because we previously assumed the prior $p(a_k)$ to be distributed uniformly on $[0,1]$, $p(a_k) = 1$ holds for any $a_k \in [0,1]$, and we obtain

$$p(a_k|n_k) = \frac{p(n_k|a_k)}{\displaystyle\int_0^1 p(n_k|s) \,\mathrm{d}s}.$$

The conditional probability $p(a_k|n_k)$ is a binomial distribution since there are either good or bad sample points. Thus, we conclude

$$p(a_k|n_k) = \frac{\binom{N}{n_k} {a_k}^{n_k} (1-a_k)^{N-n_k}}{\displaystyle\int_0^1 \binom{N}{n_k} s^{n_k} (1-s)^{N-n_k} \,\mathrm{d}s} = \frac{{a_k}^{n_k} (1-a_k)^{N-n_k}}{\displaystyle\int_0^1 s^{n_k} (1-s)^{N-n_k} \,\mathrm{d}s}.$$

The probability of the true fraction of good design space a_k in the box is hence given by the beta distribution (compare [34]). Finally, the desired probability for $a \in [a_{\text{low}}, a_{\text{up}}]$ is

$$P(a_{\text{low}} < a_k < a_{\text{up}}|n_k) = \int_{a_{\text{low}}}^{a_{\text{up}}} p(t|n_k) \,\mathrm{d}t = \frac{\displaystyle\int_{a_{\text{low}}}^{a_{\text{up}}} t^{n_k} (1-t)^{N-n_k} \,\mathrm{d}t}{\displaystyle\int_0^1 s^{n_k} (1-s)^{N-n_k} \,\mathrm{d}s}.$$

□

In [53], the 95% credible intervals for several ratios n_k^{good}/N are calculated. For $N = 100$ and in the worst case, i.e. when equally many good and bad design points are sampled and thus $n_k^{\text{good}}/N = 0.5$, the credible interval is $[a_{\text{low}}, a_{\text{up}}] \approx [0.4036, 0.5964]$ with $a_{\text{up}} - a_{\text{low}} \approx 0.2$. The larger the ratio n_k^{good}/N or $\left(N - n_k^{\text{good}}\right)/N$ is, the smaller the width of the credible interval becomes, up to $[a_{\text{low}}, a_{\text{up}}] \approx [0.9641, 0.9997]$ and $a_{\text{up}} - a_{\text{low}} \approx 0.04$ for $n_k^{\text{good}}/N = 1$. This means that, for a given credible level, we can say more precisely what the true fraction of good design space a_k likely is if the ratio n_k^{good}/N is large.

[40] is also interested in the probability of Ω_{box} containing *only* good design space, that is $a_k = 1$, if $n_k^{\text{good}}/N = 1$ and $N = 100$. Setting $a_{\text{up}} = 1$ and requiring

$$P\left(a_{\text{low}} < a_k \leq 1 \middle| n_k^{\text{good}}\right) = 0.95$$

yields $a_{\text{low}} \approx 0.97$. This means that the true fraction of good design space a_k lies in the interval $[0.97, 1]$ with a probability of 95%. Note that this credible interval is shorter than the shortest interval from above, because here

$$P\left(a_k > a_{\text{up}} \middle| n_k^{\text{good}}\right) = 0.$$

In contrast, the intervals above have been constructed with the general condition

$$P\left(a_k > a_{\text{up}} \middle| n_k^{\text{good}}\right) = 0.025.$$

From these calculations we can conclude that about $N = 100$ sample points are enough to ensure a high quality of Ω_{box} (with regards to the amount of good design space), as can be seen from Figure 2.10. More sample points increase the precision of the credible intervals only very little and may not compensate the increased complexity of the algorithm that comes with more evaluations of the objective function f. Thus, although it may improve the algorithm's speed of convergence or the overall volume of Ω_{box}, increasing the number of sample points beyond $N = 100$ with the intention to obtain more good design space within Ω_{box} is deemed inefficient. As these calculations do not require any information about Ω_{ds}, except that good and bad designs are uniformly distributed, they can be regarded as valid for all problems where sample points that are uniformly distributed have to be divided into two groups. Furthermore, they tell us that the quality (with respect to the amount of good design space) of Ω_{box} at the end of the consolidation phase must be very high. While it may be unlikely that $a_k = 1$ holds in the final step, a_k lies in $[0.97, 1]$ with a probability of 95% for $N = 100$. Thus, the probability of sampling bad designs should be very low. This is considered acceptable since the algorithm provides designers with a good first impression of the design space available. As it is usually applied in the early design process, designs will be further validated later in the design process and possible bad designs would be detected long before they become problematic.

Note that the sample size N may influence the number of steps during the exploration phase, n^{exp}, and the consolidation phase, n^{con}. If there is a cost for the evaluation of the objective function f, the total number of evaluations may perhaps not exceed a certain budget B. Thus, the total number of steps the algorithm is allowed to perform is at most B/N. In general, one can simply set $n^{\text{exp}} = n^{\text{con}} = \frac{1}{2} \cdot \frac{B}{N}$,

i.e., the exploration and consolidation phase both are allowed to perform the same number of steps. However, since the consolidation phase may terminate early if only good points are sampled three times in series, allotting more steps to the exploration phase may allow the algorithm to find a larger box before it switches to the consolidation phase which in turn may yield a larger final box. If the sample size allows the algorithm to perform only a few steps, it may not have converged on a satisfying box. In this case, the sample size should be decreased to increase the number of steps the algorithm may take.

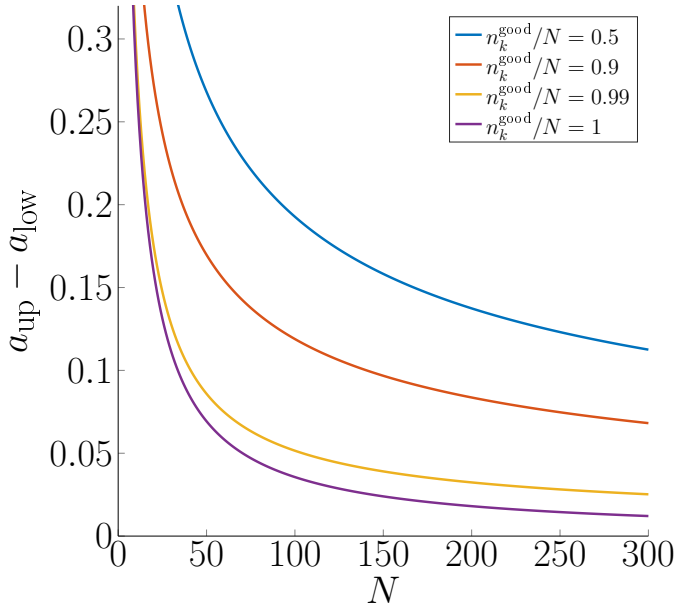

Figure 2.10: Width of the credible interval $[a_{\text{low}}, a_{\text{up}}]$ with respect to the number of sample points and the fraction n_k^{good}/N.

2.4 Numerical Experiments

In this section, we study two toy problems to demonstrate the box optimization algorithm. Additionally, we study the impact of the growth parameter on the result of the algorithm. Further numerical results pertaining to the box optimization algorithm are collected and later compared to its extensions in Chapters 5 and 6.

2.4.1 Two Example Problems in 2D

We present two simple problems in 2D to which we apply the box optimization algorithm. These problems have been analyzed in great detail in [40]. We repeat them to demonstrate the functionality of the algorithm and to convey a basic understanding of how it operates. We will also apply the other algorithms presented in the later chapters to these problems such that we can easily compare them to each other on a basic level.

2D Polygon

For

$$\mathbf{A} := \begin{bmatrix} 1/8 & 1/4 \\ 4/17 & 2/17 \\ -1/2 & 1/2 \\ -1/2 & -1/3 \\ -1/3 & -2/3 \\ 1 & -3/2 \end{bmatrix} \quad \text{and} \quad \mathbf{b} := \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ 1 \end{bmatrix},$$

we consider the problem of finding $\boldsymbol{x} \in \Omega_{\text{ds}} := [0, 4]^2$ such that

$$g(\boldsymbol{x}) := \mathbf{A}\boldsymbol{x} - \mathbf{b} \leq \mathbf{0}.$$

Note that "$\leq$" is meant entry by entry, i.e., $\boldsymbol{x} \leq \boldsymbol{y} \Leftrightarrow x_i \leq y_i$ for all i. This amounts to an objective function of the form

$$f(\boldsymbol{x}) := \begin{cases} 0, & \text{if } g(\boldsymbol{x}) \leq \mathbf{0}, \\ 1, & \text{else}, \end{cases} \tag{2.4.1}$$

with the critical value $c := 0.5$. The good design space is a two-dimensional six-sided polygon (compare Figure 2.11). In [40], the axis-parallel box with the largest volume is calculated as

$$\Omega_{\text{opt}} := [1.18, 2.85] \times [1.23, 2.58].$$

It has a normalized volume of 0.1409. In contrast, the normalized volume of the whole polygonal good design space is 0.2959. Here and in the following chapters, the normalized volume is the volume of the solution space of interest divided by the volume of the design space, i.e.

$$\frac{\mu(\Omega_{\text{box}})}{\mu(\Omega_{\text{ds}})}.$$

We use this measure for volume because it also immediately tells us what partition of Ω_{ds} is usable design space.

We execute the box optimization algorithm 100 times for this problem. Each time, the initial box lies in the center of the polygon,

$$\Omega_{\text{box}} := [1.8, 1.9] \times [2.0, 2.1].$$

The exploration and consolidation phase consist of $n^{\text{exp}} = n^{\text{con}} = 100$ steps each. During these phases, $N = 100$ design points where sampled in each step. The growth rate is dynamic with $g^{(0)} = 0.05$ and $a^{\text{target}} = 0.6$. The resulting mean of the normalized volume of the 100 final boxes, which we subsequently simply call the *mean normalized volume*, is 0.1457 with a standard deviation of 0.0077, which is very close to the optimal value. In Table 2.1, the mean interval endpoints are given. They are defined as

$$\left[\bar{a}_i, \bar{b}_i\right] := \left[\frac{1}{N}\sum_{j=1}^{N} a_{i,j}, \frac{1}{N}\sum_{j=1}^{N} b_{i,j}\right],$$

where $[a_{i,j}, b_{i,j}]$ is the interval in dimension i of the j-th result calculated by the box optimization algorithm. They admit values that are close to the interval endpoints of the optimal box, compare Figure 2.11.

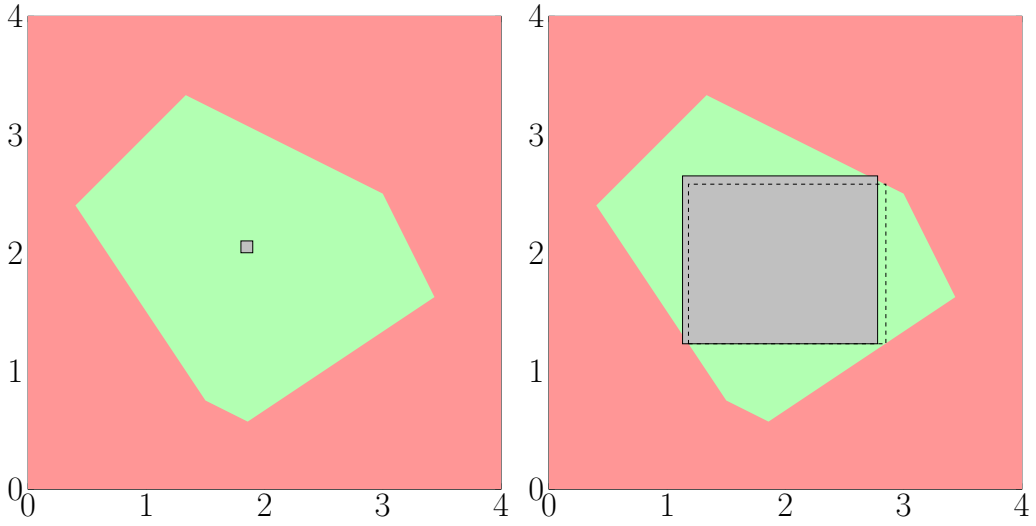

Figure 2.11: Left picture: good design space for the 2D polygon (in green) with initial box (gray). Right picture: box with the mean interval endpoints (gray) and optimal box (dashed).

	$\bar{a}_1$	$\bar{b}_1$	$\bar{a}_2$	$\bar{b}_2$
Mean	1.13	2.78	1.23	2.65
Standard Deviation	0.08	0.14	0.10	0.08

Table 2.1: Mean and standard deviation for the interval endpoints.

2D Rosenbrock Function

As our second test problem, we consider the Rosenbrock function

$$f(\boldsymbol{x}) := (1 - x_1)^2 + 100 \cdot \left(x_2 - x_1^2\right)^2 \tag{2.4.2}$$

with $\Omega_{\text{ds}} := [-2, 2] \times [-2, 3]$ and the critical value $c := 20$. The good design space admits the form of a U-shape (compare Figure 2.12) which makes this an interesting test problem. The optimal box is again calculated in [40],

$$\Omega_{\text{opt}} := [-0.525, 0.547] \times [-0.145, 0.436],$$

and has a normalized volume of 0.0311.

We execute the box optimization algorithm again 100 times with the same settings as in the previous experiment, except that the initial box is now $\Omega_{\text{box}} := [0.9, 1.1] \times [0.9, 1.1]$, containing the global minimum $\boldsymbol{x} = (1, 1)$ of f. The mean normalized volume of the final 100 boxes is 0.032 with a standard deviation of 0.0016. The mean interval endpoints are given in Table 2.2. As in the experiment before, the results are close to the optimum.

In summary, we can draw the following conclusions from these results: First, the mean normalized volumes of the boxes found by the algorithm do not deviate from the volumes of the optima by much. Second, as we can see from Figures 2.11 and 2.12 as well as Tables 2.1 and 2.2, the boxes move into a spot in the design space

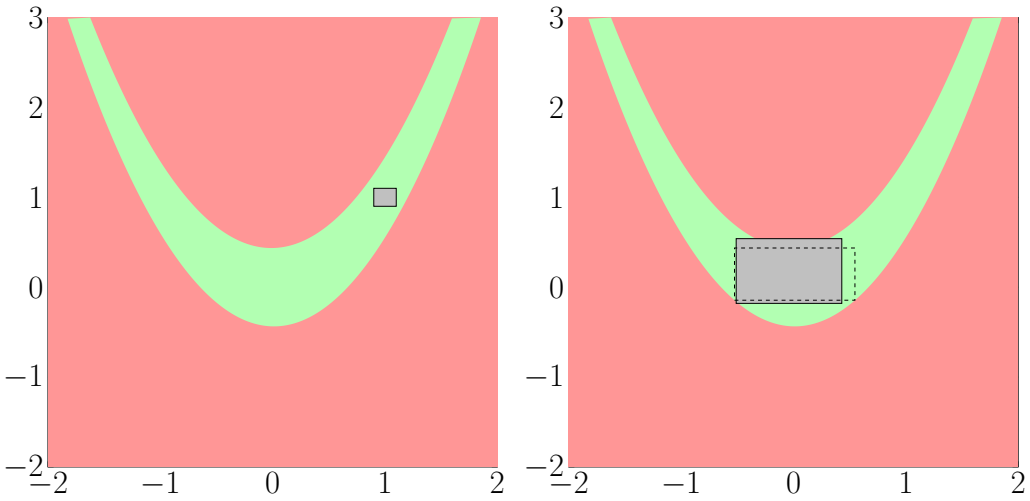

Figure 2.12: Left picture: good design space for the 2D Rosenbrock function (in green) with initial box (gray). Right picture: box with the mean interval endpoints (gray) and optimal box (dashed).

	$\bar{a}_1$	$\bar{b}_1$	$\bar{a}_2$	$\bar{b}_2$
Mean	-0.51	0.54	-0.18	0.43
Standard Deviation	0.064	0.062	0.063	0.013

Table 2.2: Mean and standard deviation for the interval endpoints.

that is close the optimum, and not somewhere else. Hence we conclude that the results are reasonable and the algorithm works as intended.

2.4.2 Growth Rate Parameter Study

While we have clarified theoretically in the previous section how the parameters N, n^{exp} and n^{con} should be chosen, the choice of the growth rate still requires a lot of knowledge about the problem at hand (see Subsection 2.3.1), which however may not be given. Thus, this section tries to give a general estimate of a feasible growth rate.

2D Rosenbrock Function

As our first test problem, we consider again the Rosenbrock function

$$f(\boldsymbol{x}) = (1 - x_1)^2 + 100 \cdot \left(x_2 - x_1^2\right)^2$$

with $\Omega_{\text{ds}} = [-2, 2] \times [-2, 3]$ and the critical value $c = 20$. For this experiment, the initial box is $\Omega_{\text{box}} := [1.54, 1.6] \times [2.44, 2.5]$, i.e., it is placed in the upper right corner of the design space (see Figure 2.13). The challenge of this experiment is that the box should traverse the right hand side of the U-shape and find the good design space at the bottom. As the good design space around the initial box approximates the

shape of a corridor delimited by almost parallel lines, the algorithm will find almost the same volume of good design space while moving the box short distances in either direction of the initial part of the corridor. Thus, the success of the algorithm will mostly depend on the growth rate.

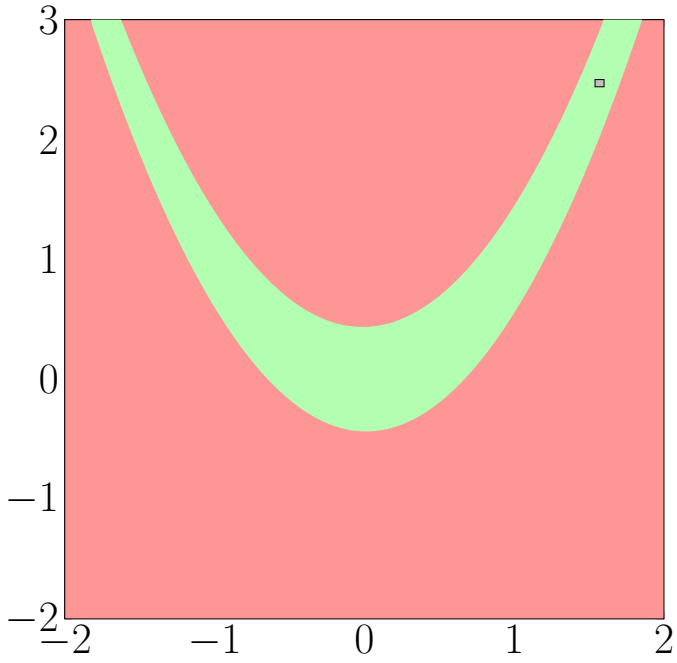

Figure 2.13: Good design space for the Rosenbrock function (in green) with the initial box in the upper right corner.

The number of steps in the exploration phase and the consolidation phase is set to $n_{\text{exp}} = n_{\text{con}} = 100$ steps, and in each step, $N = 100$ design points are sampled. The experiment is performed with constant and dynamic growth rates. For the experiments with constant growth rates, $g^{(0)}$ is set to 0.01, 0.05, 0.1, 0.2, 0.4 and 0.8. For those with dynamic growth rates, $g^{(0)} = 0.05$ and a^{target} is set to 0.5, 0.6, 0.7, 0.8, 0.9 and 1. At $a^{\text{target}} = 0.5$, we desire a balance between good and bad designs and at $a^{\text{target}} > 0.5$, we prefer good over bad designs. For $a^{\text{target}} = 1$, the algorithm will try to keep the box in the good design space. We do not set $a^{\text{target}} < 0.5$ as this would mean that we prefer sampling more bad than good designs inside the box.

For each setting of the growth rate, the box optimization algorithm is executed 100 times. Figures 2.14 and 2.15 show the resulting 100 boxes of each setting.

As can be seen from Table 2.3 and Figure 2.14, if the growth rate is constant and $g^{(0)}$ is small, e.g. 0.01, 0.05 or 0.1, the box stays near its initial position and has thus only a very small volume. However, if $g^{(0)}$ is large, e.g. 0.4 or 0.8, the algorithm finds the bottom of the U-shape more often and the resulting box has a larger volume. Curiously, for the largest value $g^{(0)} = 0.8$, the algorithm overshoots in the sense that the box is sometimes moved into the *left* corridor of the U-shape (compare Figure 2.14). This means that the algorithm moves the box into the good design space at the bottom of the U-shape but does not keep it there. Instead, it is moved into the left corridor, where the size of the box is suboptimal. We will explain this effect in the following section in more detail.

$g^{(0)}$	Mean	Standard Deviation
0.01	$0.31 \cdot 10^{-2}$	$0.02 \cdot 10^{-2}$
0.05	$0.35 \cdot 10^{-2}$	$0.02 \cdot 10^{-2}$
0.1	$0.37 \cdot 10^{-2}$	$0.04 \cdot 10^{-2}$
0.2	$0.52 \cdot 10^{-2}$	$0.48 \cdot 10^{-2}$
0.4	$2.78 \cdot 10^{-2}$	$0.96 \cdot 10^{-2}$
0.8	$\mathbf{2.87 \cdot 10^{-2}}$	$\mathbf{0.70 \cdot 10^{-2}}$

Table 2.3: Mean and standard deviation for the normalized volume of 100 boxes calculated with constant growth rates. The best result is printed in bold.

a^{target}	Mean	Standard Deviation
0.5	$\mathbf{3.18 \cdot 10^{-2}}$	$\mathbf{0.20 \cdot 10^{-2}}$
0.6	$3.03 \cdot 10^{-2}$	$0.74 \cdot 10^{-2}$
0.7	$1.55 \cdot 10^{-2}$	$1.30 \cdot 10^{-2}$
0.8	$0.42 \cdot 10^{-2}$	$0.08 \cdot 10^{-2}$
0.9	$0.36 \cdot 10^{-2}$	$0.03 \cdot 10^{-2}$
1	$0.35 \cdot 10^{-2}$	$0.02 \cdot 10^{-2}$

Table 2.4: Mean and standard deviation for the normalized volume of 100 boxes calculated with dynamic growth rates.

For a dynamic growth rate, a large value a^{target}, for example 0.8, 0.9 or 1, means that the grown box should contain mostly good design space, which results in a conservative, small growth rate. The box stays near its initial position and achieves only a small volume (see Table 2.4 and Figure 2.15). Conversely, a smaller value a^{target}, e.g. 0.5 or 0.6, results in a larger growth rate where the grown box may contain more bad design space. The resulting boxes lie in the bottom of the good design space most of the time and have thus a large volume.

In conclusion, a constant growth rate with $g^{(0)}$ between 0.4 and 0.8 or a dynamic growth rate with a^{target} about 0.5 or 0.6 seem to be good choices for this problem and may be appropriate for other problems where the good design space has a similar shape. Most importantly, these experiments confirmed the intuition that a constant growth rate with small $g^{(0)}$ or a dynamic growth rate with large a^{target} will possibly let the box explore only a small part of the design space.

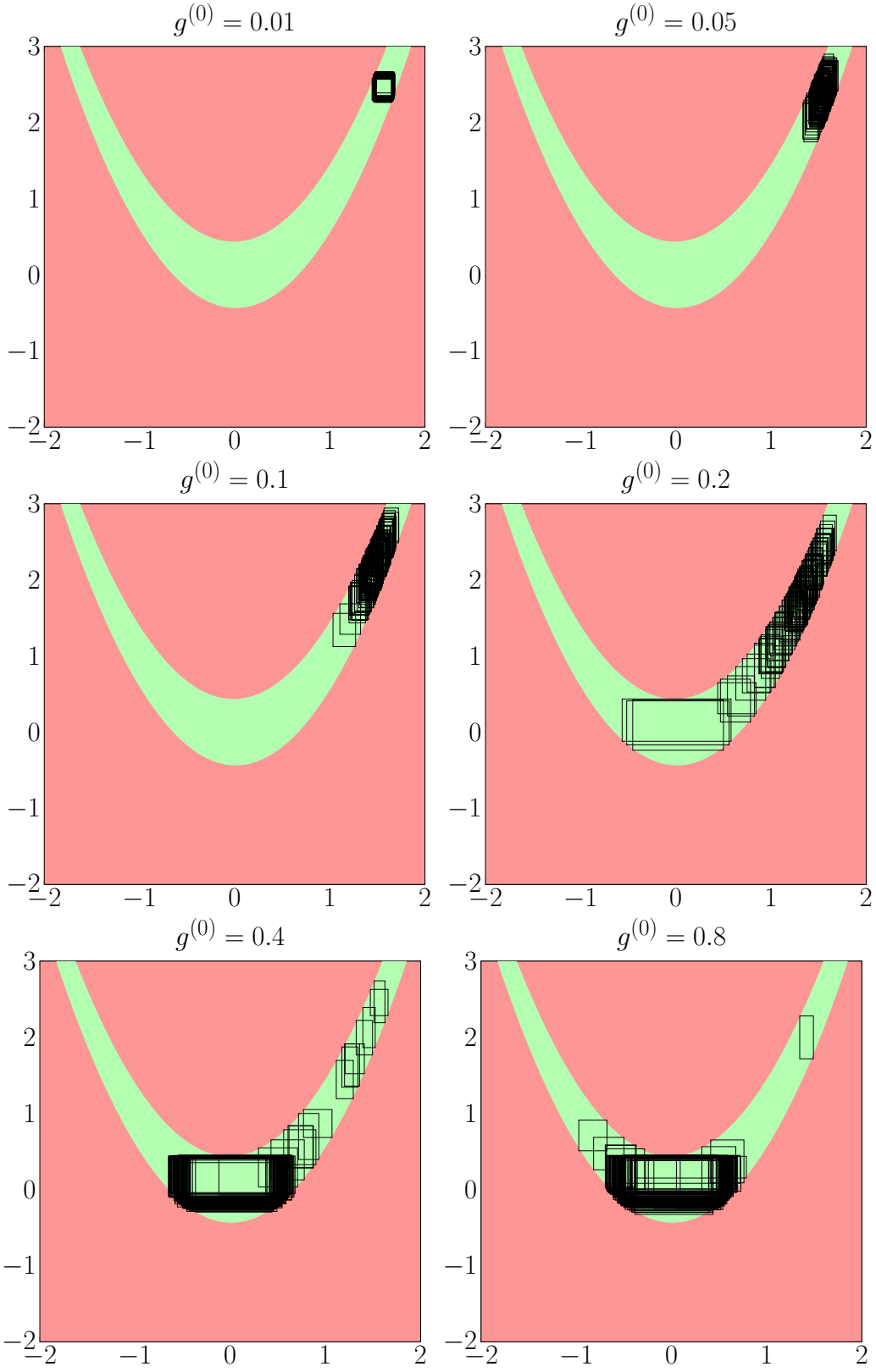

Figure 2.14: 100 boxes resulting from the box optimization algorithm with different constant growth rates.

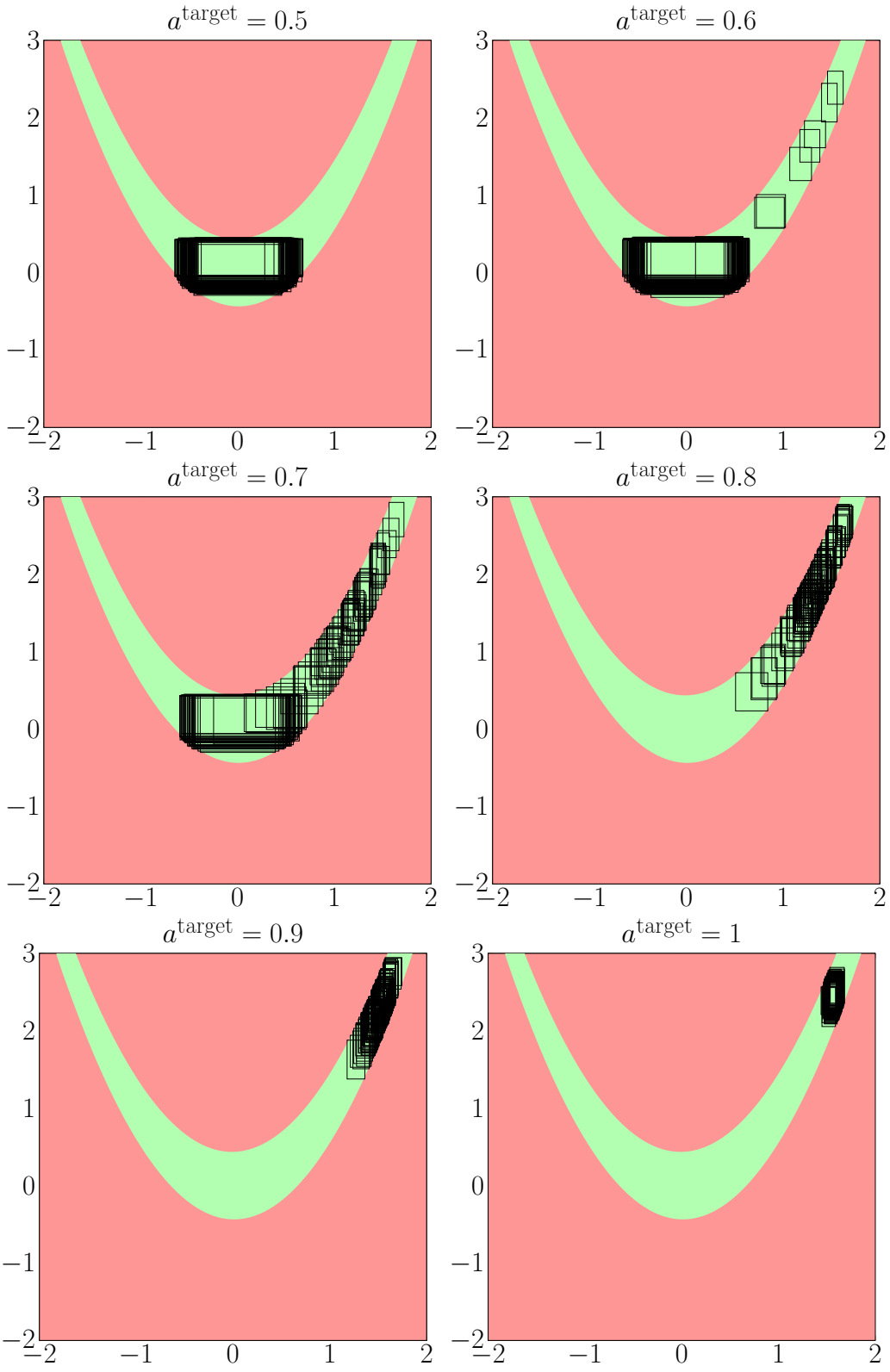

Figure 2.15: 100 boxes resulting from the box optimization algorithm with different dynamic growth rates.

8D Nonlinear Problem from Acoustics

Next, we shall consider an engineering problem from acoustics. Nine transfer paths of noise have to be designed such that the total noise generated stays below a critical value. The noise emitted by a source follows the complex-valued transfer function

$$A_k(\omega)e^{i\phi_k(\omega)}.$$

Here, ω is the frequency of the source, $A_k : [0, 2\pi] \to \mathbb{R}$ is the amplitude in the receiving region and $\phi_k(\omega) : [0, 2\pi] \to [0, 2\pi]$ the frequency in the receiving region. The total noise level is the sum of the noise generated by all emitting sources

$$|x_1 e^{ix_2} + x_3 e^{ix_4} + \ldots + x_{17} e^{ix_{18}}|. \tag{2.4.3}$$

For this experiment, the dependency on the frequency ω has been omitted, as every relevant frequency is analyzed. The amplitudes are set to the constant value $x_1 = x_3 = \ldots = x_7 = 1$, and the frequency x_2 is fixed at $x_2 = 0$. Thus, the expression from (2.4.3) yields the objective function

$$f(\boldsymbol{x}) := \left| 1 + \sum_{\ell=1}^{8} e^{ix_\ell} \right|$$

with $\Omega_{\text{ds}} := [0, 2\pi]^8$ and $c := 1.5$. The initial box is

$$\Omega_{\text{box}} := \prod_{i=1}^{8} [a_i, b_i]$$

with a_i and b_i given in Table 2.5. A visualization of the initial box can be found in Figure 2.16.

i	1	2	3	4	5	6	7	8
a_i	4.7545	4.9248	1.8186	0.4240	3.2106	1.2389	4.6416	1.2389
b_i	5.6455	5.2958	2.4098	1.1439	3.6663	1.7502	5.0973	1.7502

Table 2.5: Intervals for the initial box of the acoustics problem.

Note that the visualization is different from before. Pairs of intervals are drawn together in one *2D-map* (see also Section 3.1), i.e. the product $[a_1, b_1] \times [a_2, b_2]$ is plotted on the projection of Ω_{ds} into the dimensions 1 and 2, which we call the 2D-map $\Omega_{1,2}$, the product $[a_3, b_3] \times [a_4, b_4]$ is plotted on the projection of Ω_{ds} into the dimensions 3 and 4, which is the 2D-map $\Omega_{3,4}$, and so on. On each 2D-map $\Omega_{i,j}$, 1000 design points $\boldsymbol{x} := (x_1, \ldots, x_8)$ are sampled with $x_k \in \Omega_{\text{box}}$ for $k \neq i, j$ and $x_k \in \Omega_{i,j}$ for $k = i, j$. This means that every design point is inside of Ω_{box}, except for the coordinates x_i and x_j, which may be distributed anywhere on the 2D-map $\Omega_{i,j}$. This illustrates the region around Ω_{box} from the inside of Ω_{box}. Clearly, the initial box lies on the border of good and bad design space. The algorithm should be able to move the box into the good design space nearby.

Again, the algorithm is executed 100 times, with the same settings as in Subsection 2.4.2. The results can be found in Tables 2.6 and 2.7. Figures 2.17 and 2.18 show

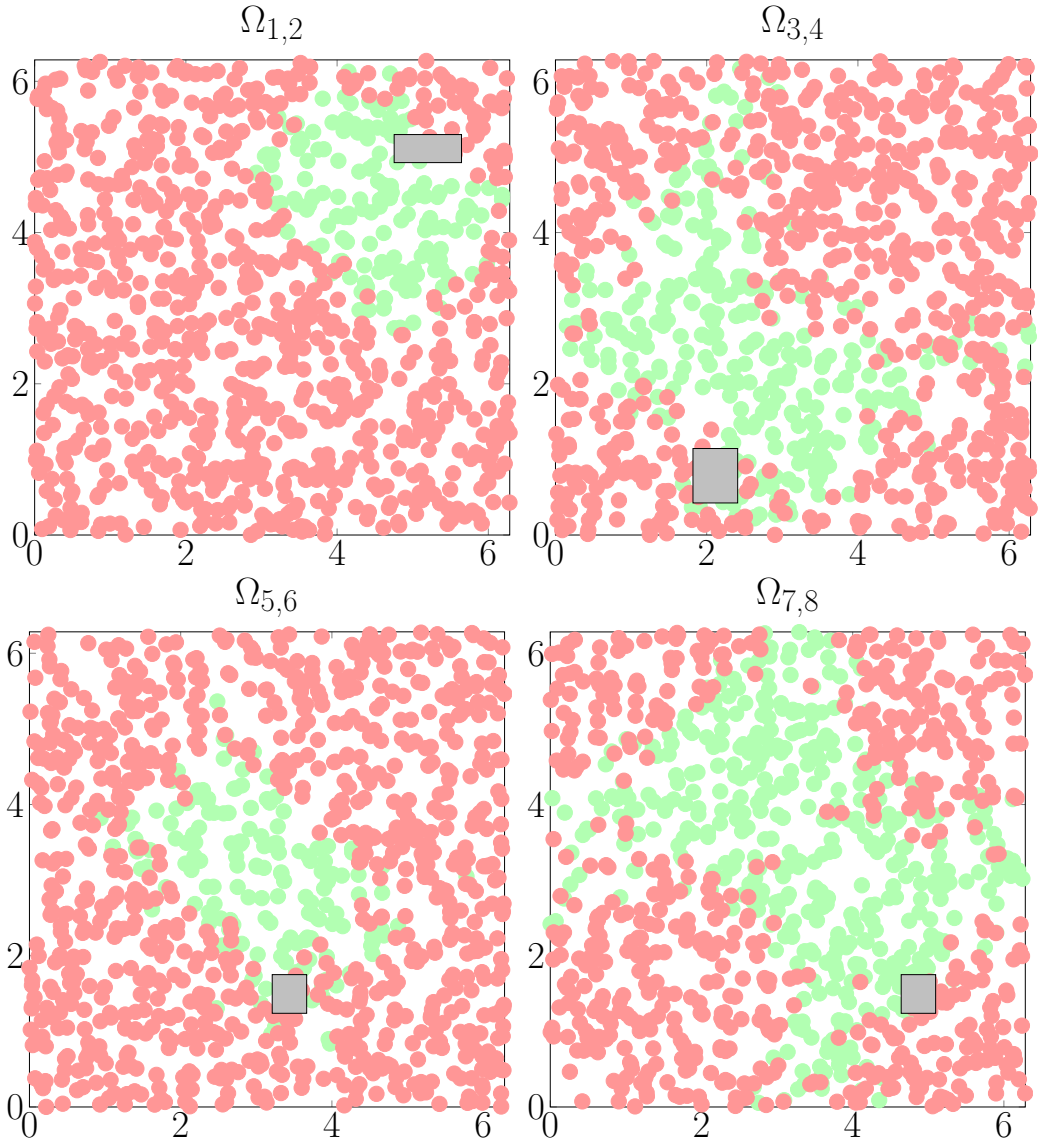

Figure 2.16: Initial box for the acoustics problem.

the final 100 boxes for each setting. The columns of these figures show the boxes on the 2D-maps $\Omega_{i,j}$ and the rows feature the different settings of $g^{(0)}$ and a^{target}. Similar to Figure 2.16, one 8D-box is represented by the product of four boxes on the four 2D-maps.

The mean normalized volume remains basically the same for all settings in the case of a dynamic growth rate. However, in the case of a constant growth rate, the mean normalized volume decreases with increasing $g^{(0)}$. This clashes with our findings from the previous section, where a larger $g^{(0)}$ also means that the final box has a larger volume. Figure 2.17 reveals the issue at hand: when the value $g^{(0)}$ is large, the final boxes are spread out over the whole design space and do not stay within the same area as it is the case for the dynamic growth rate. The reason for this is that the box from the final growth step of the exploration phase is larger than the design space, as shown in Figure 2.19. There, 100 design points are sampled such that they lie within the box and the design space. The good designs seem to be scattered

$g^{(0)}$	Mean	Standard Deviation
0.01	$0.94 \cdot 10^{-5}$	$0.37 \cdot 10^{-5}$
0.05	$\mathbf{1.05 \cdot 10^{-5}}$	$\mathbf{0.45 \cdot 10^{-5}}$
0.1	$0.71 \cdot 10^{-5}$	$0.39 \cdot 10^{-5}$
0.2	$0.51 \cdot 10^{-5}$	$0.39 \cdot 10^{-5}$
0.4	$0.28 \cdot 10^{-5}$	$0.35 \cdot 10^{-5}$
0.8	$0.20 \cdot 10^{-5}$	$0.25 \cdot 10^{-5}$

Table 2.6: Mean and standard deviation for the normalized volume of 100 boxes calculated with constant growth rates.

a^{target}	Mean	Standard Deviation
0.5	$1.06 \cdot 10^{-5}$	$0.47 \cdot 10^{-5}$
0.6	$0.99 \cdot 10^{-5}$	$0.47 \cdot 10^{-5}$
0.7	$1.05 \cdot 10^{-5}$	$0.51 \cdot 10^{-5}$
0.8	$\mathbf{1.13 \cdot 10^{-5}}$	$\mathbf{0.51 \cdot 10^{-5}}$
0.9	$0.95 \cdot 10^{-5}$	$0.38 \cdot 10^{-5}$
1	$1.02 \cdot 10^{-5}$	$0.46 \cdot 10^{-5}$

Table 2.7: Mean and standard deviation for the normalized volume of 100 boxes calculated with dynamic growth rates.

throughout the design space with no apparent pattern. In the next trimming step, the box will be trimmed such that at least one good design is contained within it, but probably not all of them. This means that the box will settle in a specific area within the design space. As this area depends heavily on the constellation of good and bad designs sampled within the box, the position of this area and, by extension, the position of the box, is random and different after each execution of the algorithm, as seen in Figure 2.17. This mechanism also explains the boxes found in the left arm of the U-shape in the bottom right picture of Figure of 2.14: The box from the final growth step of the exploration phase is very large and, by chance, good points are also sampled in the left corridor such that the optimal box can be found there.

In conclusion to the results of this study, a dynamic growth rate generally seems to be superior to a constant growth rate. For a constant growth rate, a small value $g^{(0)}$ may result in too little growth. A large $g^{(0)}$ may result in too much growth, which makes the algorithm unreliable in the sense that, with equal starting conditions, the final boxes differ much from each other, as seen in Figure 2.14. In contrast, a dynamic growth rate with a small value a^{target} yields boxes with a large volume and keeps the algorithm reliable, i.e. all boxes are similar to each other and can be found in the same area within the design space. Thus, a dynamic growth rate with $a^{\text{target}} = 0.5$ or $a^{\text{target}} = 0.6$ is a good universal setting, as it can handle good design spaces with a difficult geometry well. While it may not be optimal for all design spaces, at the very least it should provide a good starting point.

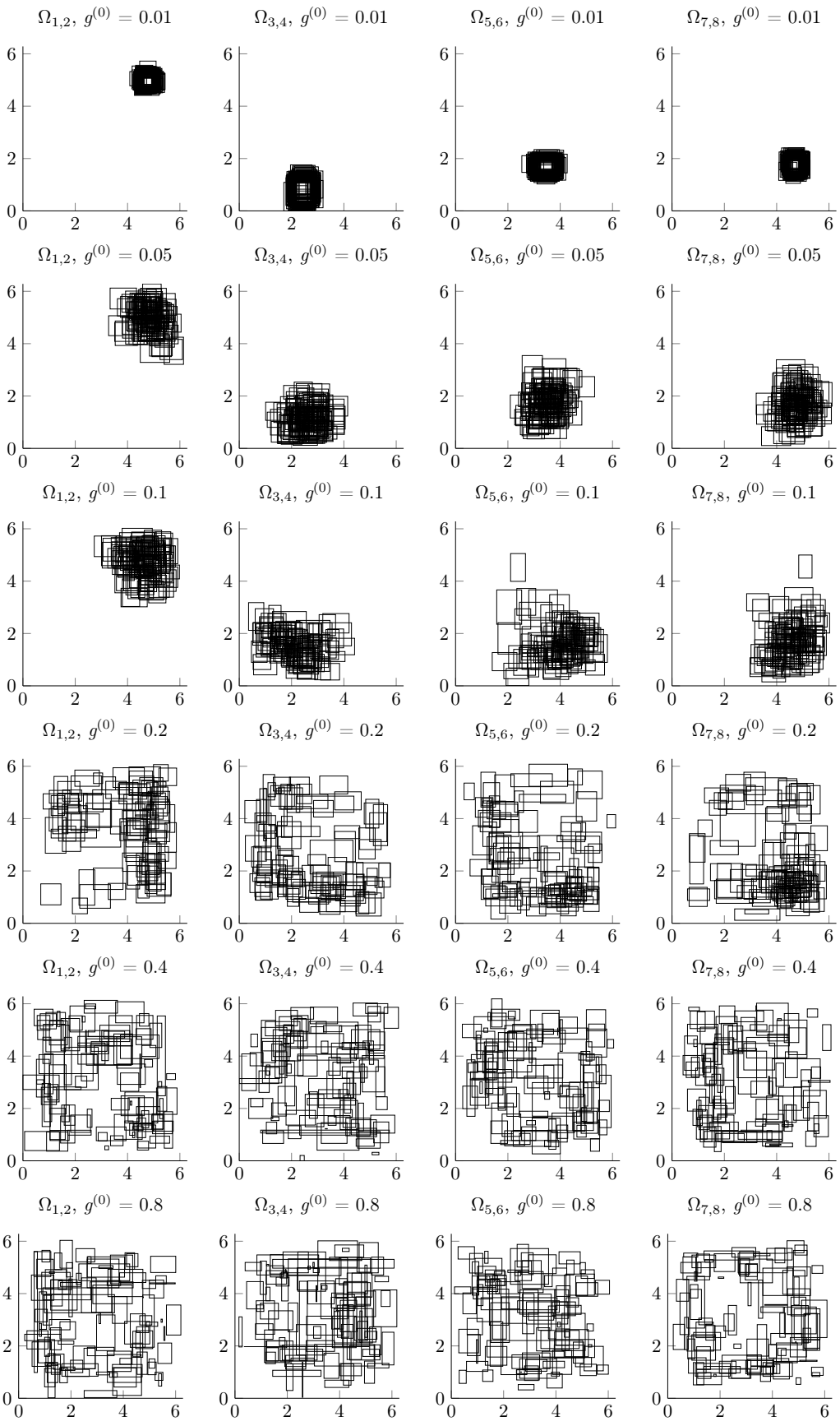

Figure 2.17: Final boxes for the acoustics problem with constant growth.

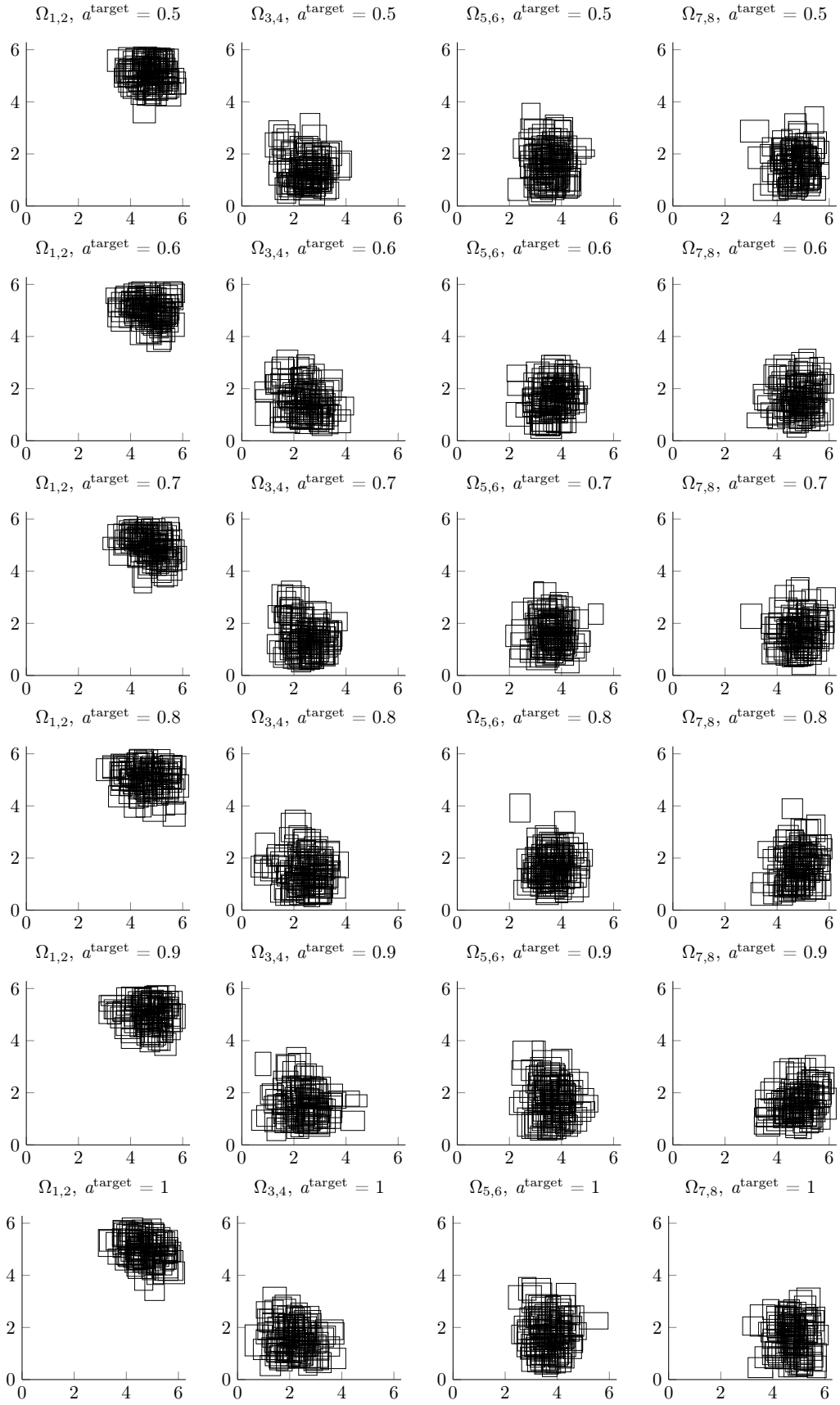

Figure 2.18: Final boxes for the acoustics problem with dynamic growth.

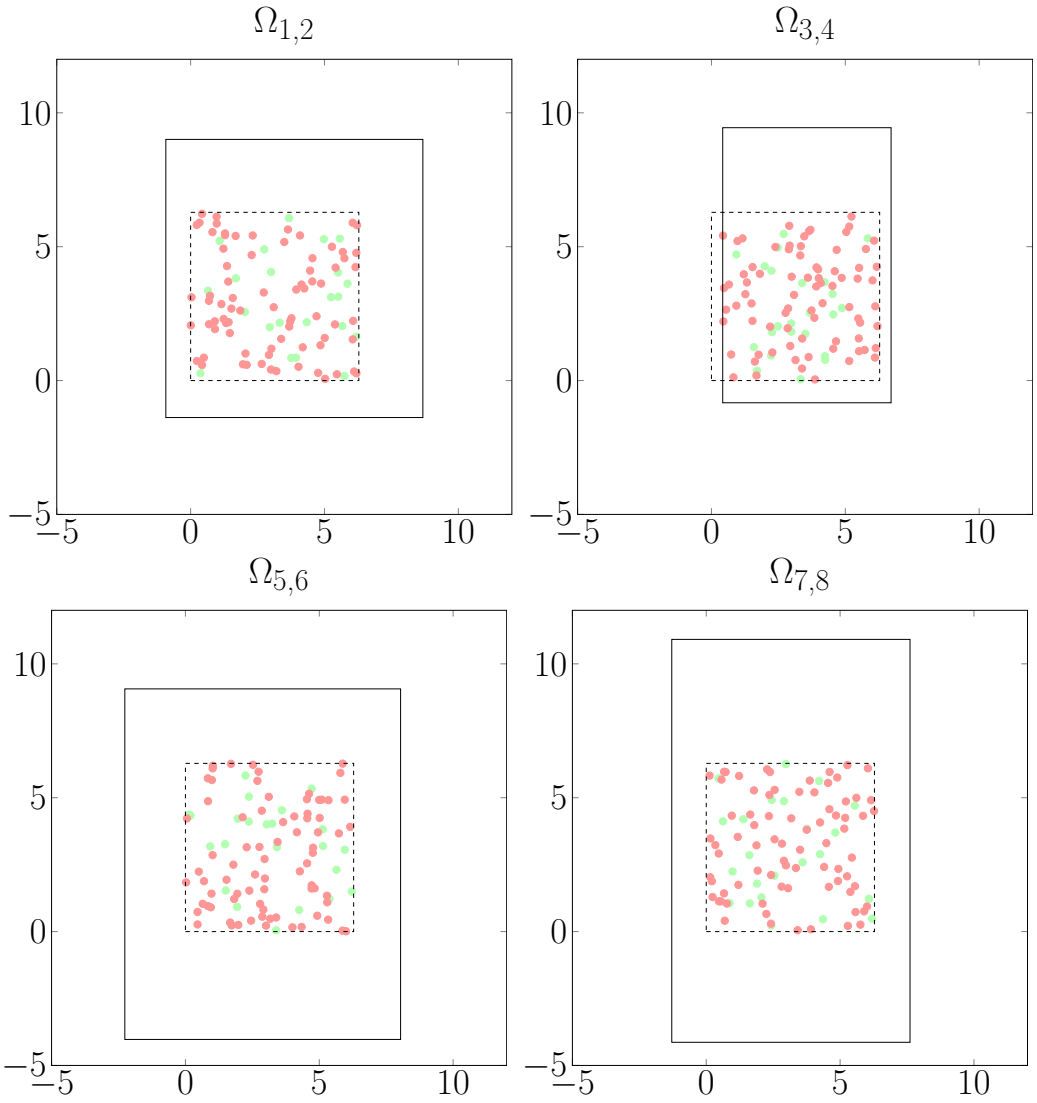

Figure 2.19: Box from the final growth step of the exploration phase for $g^{(0)} = 0.8$ with 100 designs sampled inside. The dashed line indicates Ω_{ds}.

Chapter 3

Rotated Box Optimization

This chapter presents the first extension of the box optimization algorithm, the *rotated box optimization algorithm*. It has been studied in [44]. The idea of the rotated box optimization algorithm is the following: In applications of the box optimization algorithm, often some intervals of the result are too small for practical use. For example, if the good design space Ω_c (compare Definition 1.4.2) takes the form of a diagonal strip in two dimensions (as illustrated in Figure 3.1), an axis-parallel box will be small in relation to the whole volume of Ω_c. Therefore, we modify the original problem statement to allow two-dimensional box rotations for specific pairings of design variables. On the one hand, this introduces a coupling between these pairs (however, the particular pairs remain uncoupled from the other pairs). On the other hand, this increases the solution space considerably, especially if strongly correlated design variables are taken as pairs.

3.1 Box Rotations for 2D-Maps

In order to allow for box rotations, the concept of *2D-maps* is utilized. They have been introduced as *2D-spaces* in [25], where the coupling of pairs of design variables has been studied in the case when f is a linear function.

Definition 3.1.1. *The* 2D-map $\Omega_{i,j}$ *is defined as*

$$\Omega_{i,j} := \left\{ \boldsymbol{y} \in \mathbb{R}^2 \,\middle|\, \boldsymbol{y} = \pi_{i,j}(\boldsymbol{x}),\, \boldsymbol{x} \in \Omega_{\text{ds}} \right\},$$

where $\pi_{i,j}$ *is the projection* $(x_1, \ldots, x_d) \mapsto (x_i, x_j)$ *with* $i, j = 1, \ldots, d$. *That is, the 2D-map* $\Omega_{i,j}$ *is the projection of* Ω_{ds} *onto the dimensions* i *and* j.

The rotated box optimization algorithm extends the box optimization algorithm, as explained in the following. Two design variables x_i and x_j are paired and associated with the 2D-map $\Omega_{i,j}$. The projected box $\pi_{i,j}(\Omega_{\text{box}})$ is rotated in $\Omega_{i,j}$ such that it is no longer axis-parallel in these two dimensions. This means that the box can no longer be represented by two intervals for these dimensions. Rather, it is now given by a linear combination of two vectors that are not axis-parallel (see Section 3.3). The design variables x_i and x_j are thereby coupled. However, this trade-off is acceptable since the coupling does not include the other design variables. This means that changes can be made to the design variable x_i, for example, and only x_i

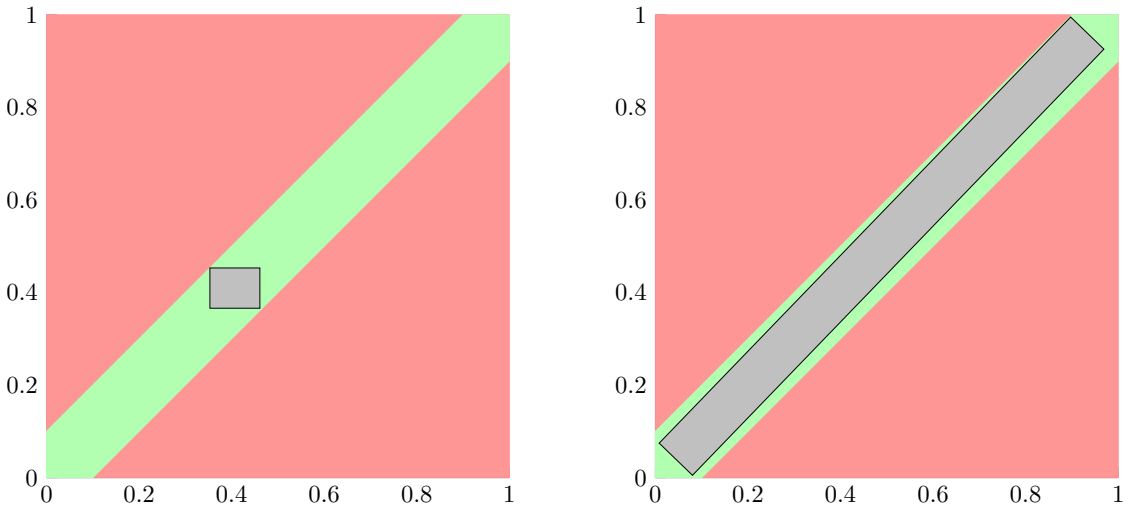

Figure 3.1: An axis-parallel box (left) and a rotated box (right) as solution spaces. Good and bad design regions are green and red, respectively.

and x_j have to be checked whether they still lie within the rotated box $\pi_{i,j}(\Omega_{\text{box}})$. The other design variables are not affected, which saves time and resources during the design process.

A meaningful choice of design variables to be coupled in a single 2D-map may be inferred from the design problem. It may make sense, e.g., to couple design variables associated with one component while keeping them uncoupled from those associated with another component. It is also reasonable to couple design variables that one designer has full access to. Each design variable is paired with at most one other design variable, such that each design variable is associated with at most one 2D-map. Some design variables may not need to be paired with any other design variables, usually because they are expected to have enough available design space. These design variables are assigned to intervals, as in the box optimization algorithm. Therefore, the box Ω_{box} is the product of one-dimensional intervals I_k and two-dimensional rotated boxes $B_{i,j}$,

$$\Omega_{\text{box}} := \prod_{k \in \mathcal{J}_I} I_k \times \prod_{(i,j) \in \mathcal{J}_{\text{pair}}} B_{i,j},$$

where

$$\mathcal{J}_I := \big\{ i \in \{1, \ldots, d\} \,\big|\, i \text{ is an unpaired dimension} \big\},$$
$$\mathcal{J}_{\text{pair}} := \big\{ (i,j) \in \mathcal{J}_B \times \mathcal{J}_B \,\big|\, \text{the dimensions } i \text{ and } j \text{ are coupled} \big\}$$

and

$$\mathcal{J}_B := \{1, \ldots, d\} \setminus \mathcal{J}_I.$$

Additionally, we define the space of all *admissible* rotated boxes as

$$\mathcal{S}_{\text{rot}} := \left\{ \Omega_{\text{box}} \subset \Omega_{\text{ds}} \;\middle|\; \Omega_{\text{box}} = \prod_{k \in \mathcal{J}_I} I_k \times \prod_{(i,j) \in \mathcal{J}_{\text{pair}}} B_{i,j} \right\}.$$

With this definition of rotated boxes at hand, we replace problem (1.4.1) by the following one:

$$\begin{cases} \text{Maximize the volume } \mu\left(\Omega_{\text{box}}\right) \\ \text{over all rotated boxes } \Omega_{\text{box}} \in \mathcal{S}_{\text{rot}} \\ \text{subject to } f(\boldsymbol{x}) \leq c \text{ for all } \boldsymbol{x} \in \Omega_{\text{box}}. \end{cases} \tag{3.1.1}$$

The goal of the rotated box algorithm is to solve this problem.

3.2 Principal Component Analysis

The most important piece of the rotated box algorithm is the *principal component analysis* (PCA, [50, 80]), also known as *singular value decomposition* (SVD, [67]), *Karhunen-Loève transform* (KLT, [35, 76]), or *proper orthogonal decomposition* (POD, [14]). PCA is applied if the dimensionality of given data should be reduced. This may be the case when one is only interested in certain characteristics or features of the data and not the whole data set itself, as, for example, in computer vision (see [76]). The feature provided by PCA is the data's variance. By applying a linear transformation to the data, it identifies directions that are orthogonal to each other and maximize the variance of the data. These directions, which are called *principal components*, form a new coordinate system where the data are spread along the coordinate axes. If we know the principal components, we can rotate the data such that they are spread parallelly along the axes of the new coordinate system (see Figure 3.2).

In [50], [51] and [80], the principal components are derived as follows. Let

$$\mathbf{X} := (X_1, \ldots, X_n)^\top$$

be a vector of random variables $X_1, \ldots, X_n$. We seek linear combinations ξ_i, $i = 1, \ldots, n$, with

$$\xi_i := \sum_{j=1}^{n} a_{i,j} X_j$$

or

$$\boldsymbol{\xi} := \mathbf{A}^\top \mathbf{X}$$

for some matrix $\mathbf{A}$ of coefficients. Note that $\boldsymbol{\xi}$ is also a vector of random variables. Let us first consider

$$\xi_1 = \sum_{j=1}^{d} a_{1,j} X_j$$

and set $\mathbf{a}_1 := (a_{1,1}, a_{1,2}, \ldots, a_{1,n})^\top$. We wish to maximize the variance of ξ_1 with respect to $\mathbf{a}_1$,

$$V[\xi_1] \to \max_{\mathbf{a}_1 \in \mathbb{R}^n}. \tag{3.2.1}$$

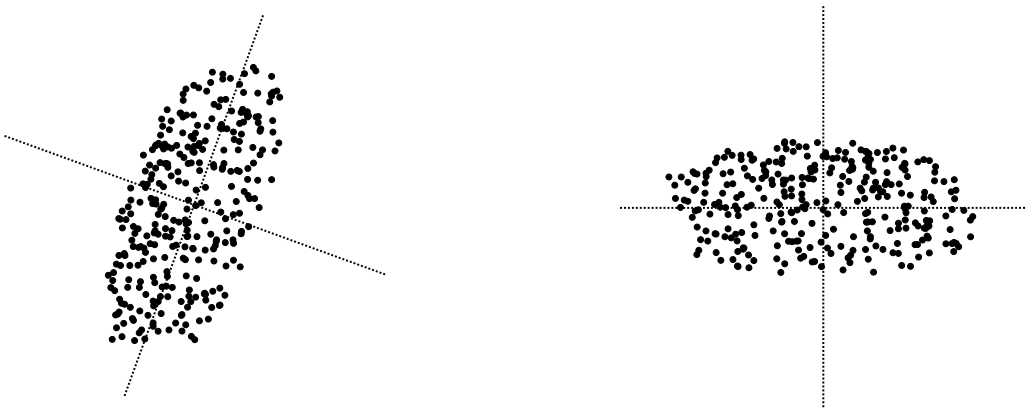

Figure 3.2: A data set with two principal components (dotted) before (left) and after (right) rotation.

However, as this maximum cannot be attained for a vector $\mathbf{a}_1$ with finite values, we add the constraint that $\mathbf{a}_1$ should be normalized, i.e.

$$\|\mathbf{a}_1\|_2^2 = \mathbf{a}_1^\top \mathbf{a}_1 = 1. \tag{3.2.2}$$

For the variance of ξ_1, it holds

$$\begin{aligned} V[\xi_1] &= E[\xi_1^2] - E[\xi_1]^2 \\ &= E\big[\mathbf{a}_1^\top \mathbf{X}\mathbf{X}^\top \mathbf{a}_1\big] - E\big[\mathbf{a}_1^\top \mathbf{X}\big] E\big[\mathbf{X}^\top \mathbf{a}_1\big] \\ &= \mathbf{a}_1^\top \Big(E\big[\mathbf{X}\mathbf{X}^\top\big] - E\big[\mathbf{X}\big] E\big[\mathbf{X}^\top\big] \Big) \mathbf{a}_1 \\ &= \mathbf{a}_1^\top \boldsymbol{\Sigma} \mathbf{a}_1, \end{aligned}$$

where $\boldsymbol{\Sigma}$ is the covariance matrix of $\mathbf{X}$.

Now we apply the method of *Lagrange multipliers* (see [49]) to maximize (3.2.1) with respect to (3.2.2). Thus, we intend to maximize

$$\mathcal{L}_\lambda(\mathbf{a}_1) := \mathbf{a}_1^\top \boldsymbol{\Sigma} \mathbf{a}_1 - \lambda\big(\mathbf{a}_1^\top \mathbf{a}_1 - 1\big).$$

Differentiation with respect to $\mathbf{a}_1$ yields

$$\mathcal{L}'_\lambda(\mathbf{a}_1) = 2\boldsymbol{\Sigma}\mathbf{a}_1 - 2\lambda\mathbf{a}_1.$$

This implies that

$$\boldsymbol{\Sigma}\mathbf{a}_1 - \lambda\mathbf{a}_1 = \mathbf{0}$$

must hold for the maximum. Because we need $\mathbf{a}_1 \neq \mathbf{0}$, λ has thus to be an eigenvalue of $\boldsymbol{\Sigma}$ and $\mathbf{a}_1$ its associated eigenvector. Since we want to maximize the variance

$$V[\xi_1] = \mathbf{a}_1^\top \boldsymbol{\Sigma} \mathbf{a}_1 = \mathbf{a}_1^\top \lambda \mathbf{a}_1 = \lambda \mathbf{a}_1^\top \mathbf{a}_1 = \lambda,$$

λ is chosen to be the largest eigenvalue. If we order the eigenvalues such that $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n \geq 0$, then $\mathbf{a}_1$ is the eigenvector to the largest eigenvalue λ_1.

For the second principal component $\xi_2 = \mathbf{a}_2^\top \mathbf{X}$, we have to find coefficients $a_{2,i}$, $i = 1, \ldots, n$, such that $V[\xi_2]$ is maximized,

$$V[\xi_2] \to \max_{\mathbf{a}_2 \in \mathbb{R}^n}$$

under the constraints that

$$\|\mathbf{a}_2\|_2^2 = \mathbf{a}_2^\top \mathbf{a}_2 = 1$$

and that ξ_2 is uncorrelated with the already known first principal component ξ_1, i.e.

$$\operatorname{cov}(\xi_1, \xi_2) = 0, \tag{3.2.3}$$

for the covariance function $\operatorname{cov}(\cdot\,, \cdot)$. Since

$$\begin{aligned}
\operatorname{cov}(\xi_1, \xi_2) &= \operatorname{cov}\big(\mathbf{a}_1^\top \mathbf{X}, \mathbf{a}_2^\top \mathbf{X}\big) \\
&= \mathbf{a}_1^\top \operatorname{cov}(\mathbf{X}, \mathbf{X}) \mathbf{a}_2 \\
&= \mathbf{a}_1^\top \boldsymbol{\Sigma} \mathbf{a}_2 = \mathbf{a}_2^\top \boldsymbol{\Sigma} \mathbf{a}_1 \\
&= \mathbf{a}_2^\top \lambda_1 \mathbf{a}_1 \\
&= \lambda_1 \mathbf{a}_2^\top \mathbf{a}_1 = \lambda_1 \mathbf{a}_1^\top \mathbf{a}_2,
\end{aligned}$$

the constraint in equation (3.2.3) can be replaced by any one of the following equations:

$$\begin{aligned}
\mathbf{a}_1^\top \boldsymbol{\Sigma} \mathbf{a}_2 &= 0, \\
\mathbf{a}_2^\top \boldsymbol{\Sigma} \mathbf{a}_1 &= 0, \\
\mathbf{a}_1^\top \mathbf{a}_2 &= 0, \\
\mathbf{a}_2^\top \mathbf{a}_1 &= 0.
\end{aligned}$$

We choose the last of the above equations such that the Lagrange function corresponding to maximize $V[\xi_2]$ under the given constraints is

$$\mathcal{L}_{\lambda,\phi}(\mathbf{a}_2) := \mathbf{a}_2^\top \boldsymbol{\Sigma} \mathbf{a}_2 - \lambda\big(\mathbf{a}_2^\top \mathbf{a}_2 - 1\big) - \phi \mathbf{a}_2^\top \mathbf{a}_1$$

with the Lagrange multipliers λ and ϕ. Differentiating with respect to $\mathbf{a}_2$ yields

$$\mathcal{L}'_{\lambda,\phi}(\mathbf{a}_2) = 2\boldsymbol{\Sigma}\mathbf{a}_2 - 2\lambda \mathbf{a}_2 - \phi \mathbf{a}_1 \stackrel{!}{=} \mathbf{0}.$$

By multiplying with $\mathbf{a}_1^\top$, we get

$$0 = 2 \underbrace{\mathbf{a}_1^\top \boldsymbol{\Sigma} \mathbf{a}_2}_{=0} - 2\lambda \underbrace{\mathbf{a}_1^\top \mathbf{a}_2}_{=0} - \phi \underbrace{\mathbf{a}_1^\top \mathbf{a}_1}_{=1} = \phi.$$

Thus, it holds

$$\boldsymbol{\Sigma}\mathbf{a}_2 - \lambda \mathbf{a}_2 = \mathbf{0},$$

which means that $\mathbf{a}_2$ must be an eigenvector of $\boldsymbol{\Sigma}$ and λ its associated eigenvalue, as before. However, λ may not be equal to λ_1, because otherwise $\mathbf{a}_2 = \mathbf{a}_1$ and the equation $\mathbf{a}_2^\top \mathbf{a}_1 = 0$ would no longer be true. Hence, in order to maximize $V[\xi_2]$,

we find that $\lambda = \lambda_2$, the second largest eigenvalue, and $\mathbf{a}_2$ is the corresponding eigenvector.

The k-th principal component is derived inductively, assuming the previous $k-1$ principal components are the eigenvectors corresponding to the $k-1$ largest eigenvalues. We wish to maximize $V[\xi_k] = \mathbf{a}_k^\top \boldsymbol{\Sigma} \mathbf{a}_k$, i.e.,

$$V[\xi_k] \to \max_{\mathbf{a}_k \in \mathbb{R}^n},$$

with the constraints

$$\|\mathbf{a}_k\|_2^2 = \mathbf{a}_k^\top \mathbf{a}_k = 1$$

and

$$0 = \operatorname{cov}(\xi_i, \xi_k) = \mathbf{a}_i^\top \boldsymbol{\Sigma} \mathbf{a}_k = \mathbf{a}_k^\top \boldsymbol{\Sigma} \mathbf{a}_i = \lambda_i \mathbf{a}_k^\top \mathbf{a}_i, \quad i = 1, \dots, k-1.$$

As before, the Lagrange function is

$$\mathcal{L}_{\lambda,\boldsymbol{\phi}}(\mathbf{a}_k) := \mathbf{a}_k^\top \boldsymbol{\Sigma} \mathbf{a}_k - \lambda\big(\mathbf{a}_k^\top \mathbf{a}_k - 1\big) - \sum_{j=1}^{k-1} \phi_j \mathbf{a}_k^\top \mathbf{a}_j.$$

Again, differentiating with respect to $\mathbf{a}_k$ yields

$$\mathcal{L}'_{\lambda,\boldsymbol{\phi}}(\mathbf{a}_k) = 2\boldsymbol{\Sigma}\mathbf{a}_k - 2\lambda \mathbf{a}_k - \sum_{j=1}^{k-1} \phi_j \mathbf{a}_j \stackrel{!}{=} \mathbf{0}.$$

Multiplying with $\mathbf{a}_i^\top$ for $i = 1, \dots, k-1$ gives

$$0 = 2\underbrace{\mathbf{a}_i^\top \boldsymbol{\Sigma} \mathbf{a}_k}_{=0} - 2\lambda \underbrace{\mathbf{a}_i^\top \mathbf{a}_k}_{=0} - \sum_{j=1}^{k-1} \phi_j \underbrace{\mathbf{a}_i^\top \mathbf{a}_j}_{=\delta_{i,j}} = \phi_i.$$

Therefore, it holds again

$$\boldsymbol{\Sigma}\mathbf{a}_k - \lambda \mathbf{a}_k = \mathbf{0}$$

and the k-th principal component is the eigenvector $\mathbf{a}_k$ to the eigenvector λ. Because we cannot choose one of the previous $k-1$ largest eigenvalues, as otherwise $\mathbf{a}_k^\top \mathbf{a}_i = 0$ would no longer hold for some i with $i = 1, \dots, n$, we find $\lambda = \lambda_k$, the k-th largest eigenvalue, and $\mathbf{a}_k$ is the corresponding eigenvector.

We have finally derived all n possible principal components for a random vector $\mathbf{X} = (X_1, \dots, X_n)^\top$. Note that, in applications, only the first m principal components with $m \ll n$ are considered, since those are associated with the most prevalent correlations. For the purpose of the rotated box algorithm, we replace each random variable X_i with a sampled design point and consider the design to be a realization of that vector. Additionally, we will only consider the first two principal components each, as we project our data to a 2D-map before applying the PCA.

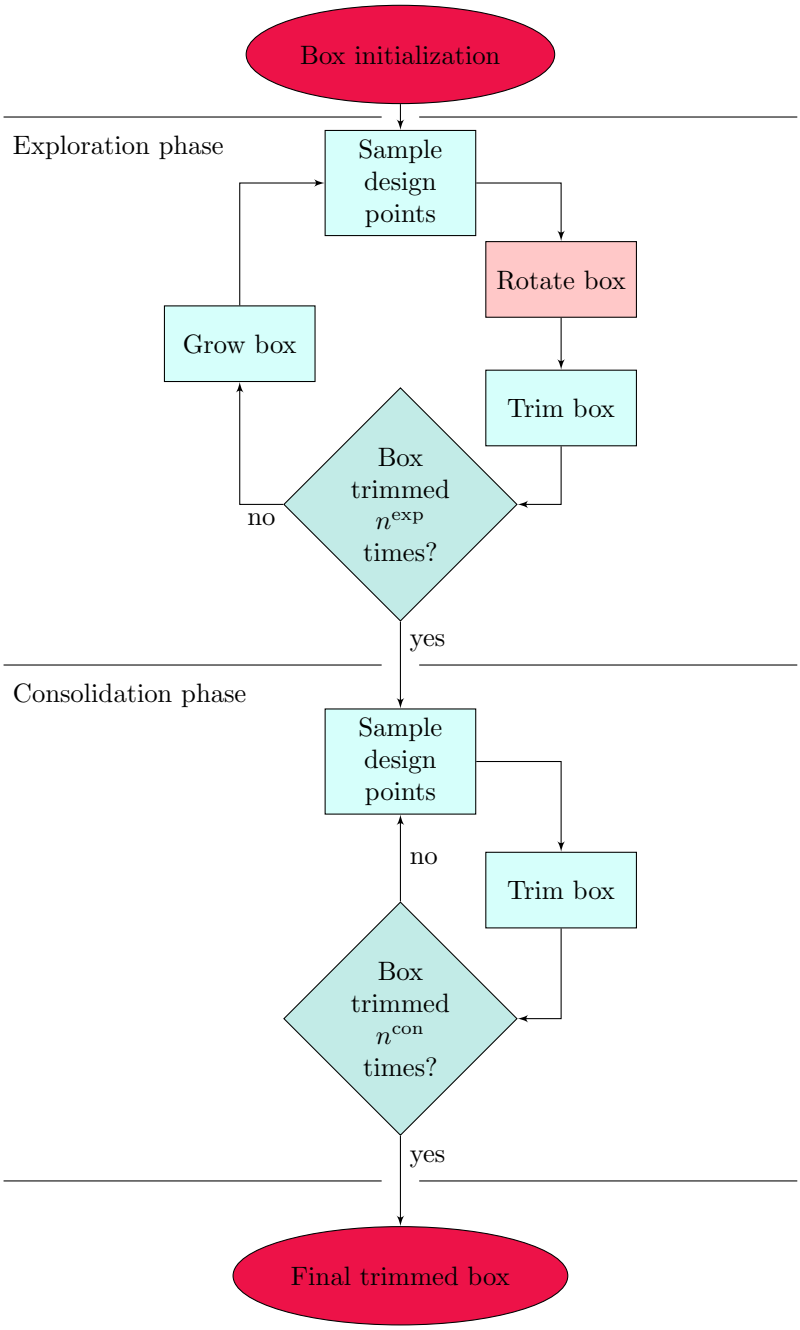

Figure 3.3: Flowchart of the rotated box optimization algorithm. It coincides with the box optimization algorithm if the field "Rotate box", highlighted in light red, would be removed.

3.3 Rotated Box Optimization Algorithm

The rotated box optimization algorithm is based on the same steps as the box optimization algorithm and extends it by one step to rotate the candidate box. "Rotate Box" is performed after sampling the design points and before trimming. The flowchart in Figure 3.3 gives an overview of the rotated box algorithm, where the "Rotate Box" step is highlighted by the light red background colour. Most steps had to be modified to account for the no longer axis-parallel boxes in the algorithm. A detailed explanation of all the steps is provided in the following subsections.

During the execution of the algorithm, Ω_{box} is represented as a vector-matrix pair $(\mathbf{V}, \mathbf{M})$. The vector $\mathbf{V} \in \mathbb{R}^d$ denotes an arbitrary vertex of the box and acts as origin of the rotated coordinate system described by the edges of the box. The matrix $\mathbf{M} := [\mathbf{m}_1, \ldots, \mathbf{m}_d] \in \mathbb{R}^{d \times d}$ contains the basis of this rotated coordinate system. Each column $\mathbf{m}_1, \ldots, \mathbf{m}_d$ of $\mathbf{M}$ is the distance vector from $\mathbf{V}$ to one of its neighboring vertices. They are ordered such that the edge that is axis-parallel in dimension i occupies the i-th column and the two edges that are associated with a 2D-map $\Omega_{i,j}$ occupy the i-th and j-th column (see the example below). This representation of rotated boxes turned out to be very useful for the following reasons:

- All other vertices of the box can be constructed by a linear combination of $\mathbf{V}$ and the columns of $\mathbf{M}$.
- Representing a d-dimensional rotated box by its vertices would require a list with 2^d entries. For large d, this would be infeasible. For example, for $d = 100$ spatial dimensions, we would need about $2^{100} \cdot 100 \cdot 64$ Bit $= 10^{23}$ Gigabyte of disk space to store the 2^{100} vertices.
- Any affine transformation of the rotated box can be carried out by applying that transformation to $\mathbf{V}$ and $\mathbf{M}$. Especially, for the rotation associated with the principal components in the dimensions i and j, defined as

  $$\rho_{i,j}(\boldsymbol{x}) := \mathbf{A}_{i,j}\boldsymbol{x}, \tag{3.3.1}$$

  the rotated coordinates can be calculated easily.
- The intervals I_k and two-dimensional rotated boxes $B_{i,j}$ can be retrieved by setting

  $$I_k := \left\{ x \in \mathbb{R} \,\middle|\, x = v_k + t \cdot m_{k,k},\, t \in [0,1] \right\}$$

  and

  $$B_{i,j} := \left\{ \boldsymbol{x} \in \mathbb{R}^2 \,\middle|\, \boldsymbol{x} = \begin{bmatrix} v_i \\ v_j \end{bmatrix} + s \cdot \begin{bmatrix} m_{i,i} \\ m_{i,j} \end{bmatrix} + t \cdot \begin{bmatrix} m_{j,i} \\ m_{j,j} \end{bmatrix},\, (s,t) \in [0,1]^2 \right\}.$$

For example, the rotated box given by the vertices $\left(4,2,0\right)$, $\left(4,4,\frac{4}{3}\right)$, $\left(4,2,\frac{13}{3}\right)$, $\left(4,0,3\right)$, $\left(0,2,0\right)$, $\left(0,4,\frac{4}{3}\right)$, $\left(0,2,\frac{13}{3}\right)$, and $\left(0,0,3\right)$, compare Figure 3.4, can be expressed by

$$\mathbf{V} = \begin{bmatrix} 4 \\ 0 \\ 3 \end{bmatrix}, \quad \mathbf{M} = \begin{bmatrix} -4 & 0 & 0 \\ 0 & 2 & 2 \\ 0 & \frac{4}{3} & -3 \end{bmatrix}.$$

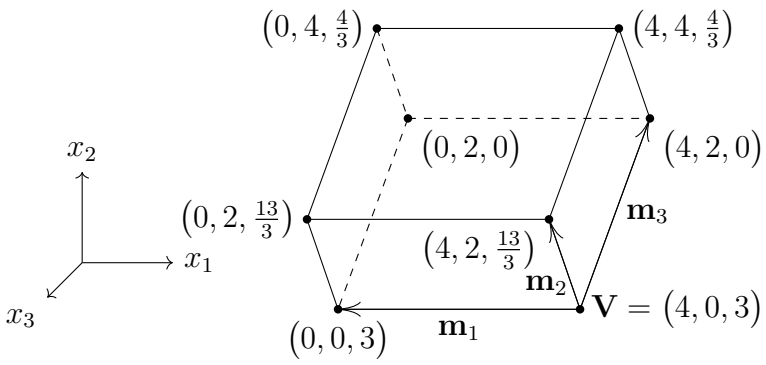

Figure 3.4: A rotated box in three dimensions.

3.3.1 Box Initialization

The box initialization step is the same as in the box optimization algorithm, compare Subsection 2.1.2. This means that either the box is given by the problem or it is constructed around an optimal design point.

3.3.2 Exploration Phase

The exploration phase is extended by the "Rotate Box" step. Now, the box is moved through the design space not only by being trimmed and grown but also by being rotated.

Sample Design Points

The step "Sample Design Points" has to be adjusted such that design points can be sampled within a rotated box Ω_{box}. However, this adjustment can be carried out easily: each design $\boldsymbol{x}$ can be sampled randomly in $[0, 1]^d$ and is then simply mapped into Ω_{box} via

$$\boldsymbol{x} \mapsto \mathbf{M}\boldsymbol{x} + \mathbf{V}.$$

After this transformation, it has to be checked whether the design point still lies within Ω_{ds} or not, as explained in Section 2.2.

Rotate Box

The optimal rotated box is determined with the help of the PCA (see Section 3.2) applied to the good design points $\mathcal{X}^{\text{good}}$, projected onto the respective 2D-map $\Omega_{i,j}$. Namely, the coordinate system of $\Omega_{i,j}$ is rotated such that the two dominant principle components form the coordinate axes (see Figure 3.5). Ω_{box} and all design points are transformed into the new coordinate systems. This is done such that the origin of the rotated coordinate system is the center of gravity of the good design points $\mathcal{X}^{\text{good}}$.

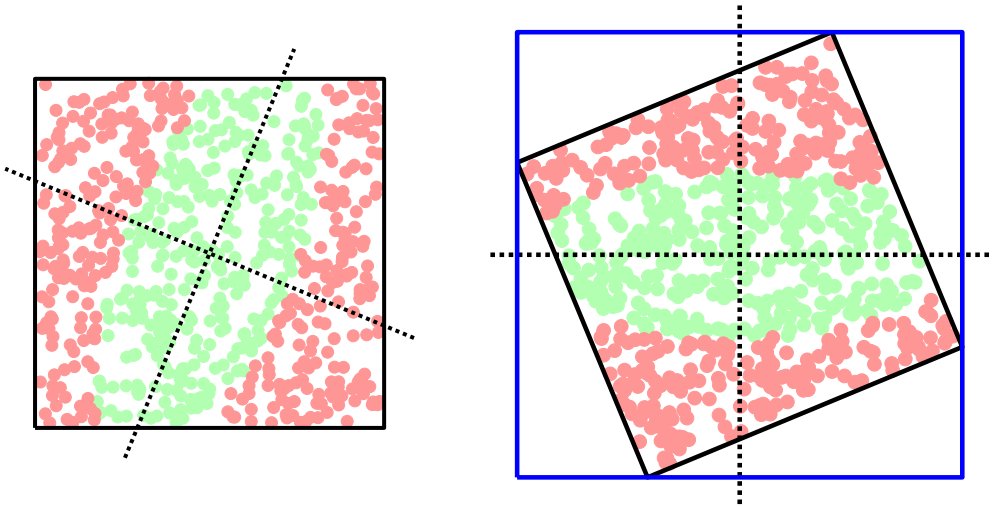

Figure 3.5: The design points and principal components (dotted) before (left) and after (right) rotation. The blue line indicates the axis-parallel bounding box around the rotated box.

The details of the box rotation step are shown in Algorithm 3. The first input parameter (see line 1) is $\Omega_{\text{box}} := (\mathbf{V}, \mathbf{M})$. The second and third inputs are two matrices

$$\mathbf{X}^{\text{good}} := \left[{}_{(1)}\boldsymbol{x}^{\text{good}}, \ldots, {}_{(n^{\text{good}})}\boldsymbol{x}^{\text{good}}\right] \in \mathbb{R}^{n^{\text{good}} \times d},$$
$$\mathbf{X}^{\text{bad}} := \left[{}_{(1)}\boldsymbol{x}^{\text{bad}}, \ldots, {}_{(n^{\text{bad}})}\boldsymbol{x}^{\text{bad}}\right] \in \mathbb{R}^{n^{\text{bad}} \times d}.$$

These are the sets $\mathcal{X}^{\text{good}}$ and $\mathcal{X}^{\text{bad}}$ written as matrices, where each row contains one design point.

The algorithm begins by iterating over the 2D-maps $\Omega_{i,j}$, see line 3 of Algorithm 3. On each 2D-map, we calculate the principal components of the good design points, which are the eigenvectors of the sample covariance matrix of the good design points on that map (line 4). The procedure `eigenvectors` calculates the *normalized* eigenvectors $\mathbf{E} := [\mathbf{e}_1, \mathbf{e}_2]$ of this 2×2 covariance matrix. For the coefficients

$$\begin{bmatrix} a & b \\ b & c \end{bmatrix} := \begin{bmatrix} \operatorname{cov}\left(\mathbf{X}_i^{\text{good}}, \mathbf{X}_i^{\text{good}}\right) & \operatorname{cov}\left(\mathbf{X}_i^{\text{good}}, \mathbf{X}_j^{\text{good}}\right) \\ \operatorname{cov}\left(\mathbf{X}_j^{\text{good}}, \mathbf{X}_i^{\text{good}}\right) & \operatorname{cov}\left(\mathbf{X}_j^{\text{good}}, \mathbf{X}_j^{\text{good}}\right) \end{bmatrix},$$

these eigenvectors are

$$\mathbf{E} := \left[\frac{\widehat{\mathbf{e}}_1}{\|\widehat{\mathbf{e}}_1\|}, \frac{\widehat{\mathbf{e}}_2}{\|\widehat{\mathbf{e}}_2\|}\right],$$

where

$$\lambda_{1,2} := \frac{a + c \pm \sqrt{(a+c)^2 - 4(a - b^2)}}{2} \quad \text{and} \quad \widehat{\mathbf{e}}_{1,2} := \begin{bmatrix} 1 \\ \frac{\lambda_{1,2} - a}{b} \end{bmatrix}.$$

In line 5 of Algorithm 3, the mean μ_i and μ_j of the good design points in the dimensions i and j are calculated. Subsequently, in lines 6 and 7, the actual rotation of the design points happens. This means that μ_i and μ_j are subtracted from all design points to normalize them with respect to the origin. Then, the design points

are multiplied with the matrix $\mathbf{E}$ to rotate them such that the principal components form the new coordinate axes. Finally, in lines 8 and 9, the same rotation is applied to the respective values of the tuple $(\mathbf{V}, \mathbf{M})$, which determines the solution space Ω_{box} in the dimensions i and j.

Note that we can now redefine the box rotation on the 2D-map $\Omega_{i,j}$ from equation (3.3.1) as

$$\rho_{i,j} : \Omega_{i,j} \to \mathbb{R}^2,\ \boldsymbol{x} \mapsto \mathbf{E} \begin{bmatrix} x_1 - \mu_i \\ x_2 - \mu_j \end{bmatrix}.$$

Algorithm 3 (Box Rotation). This algorithm rotates the box by rotating the coordinate system.

1: **Input:** Ω_{box}, $\mathbf{X}^{\text{good}}$, $\mathbf{X}^{\text{bad}}$
2: **Output:** Ω_{box}, $\mathbf{X}^{\text{good}}$, $\mathbf{X}^{\text{bad}}$

3: **for all** 2D-maps $\Omega_{i,j}$ **do**
4: $\quad \mathbf{E} \leftarrow \texttt{eigenvectors}\left(\begin{bmatrix} \text{cov}(\mathbf{X}_i^{\text{good}}, \mathbf{X}_i^{\text{good}}) & \text{cov}(\mathbf{X}_i^{\text{good}}, \mathbf{X}_j^{\text{good}}) \\ \text{cov}(\mathbf{X}_j^{\text{good}}, \mathbf{X}_i^{\text{good}}) & \text{cov}(\mathbf{X}_j^{\text{good}}, \mathbf{X}_j^{\text{good}}) \end{bmatrix} \right)$
5: $\quad [\mu_i, \mu_j] \leftarrow \frac{1}{n^{\text{good}}} \sum_{k=1}^{n^{\text{good}}} [x_{k,i}, x_{k,j}]$
6: $\quad \left[\mathbf{X}_i^{\text{good}}, \mathbf{X}_j^{\text{good}}\right] \leftarrow \left[\mathbf{X}_i^{\text{good}} - \mu_i, \mathbf{X}_j^{\text{good}} - \mu_j\right] \cdot \mathbf{E}^\top$
7: $\quad \left[\mathbf{X}_i^{\text{bad}}, \mathbf{X}_j^{\text{bad}}\right] \leftarrow \left[\mathbf{X}_i^{\text{bad}} - \mu_i, \mathbf{X}_j^{\text{bad}} - \mu_j\right] \cdot \mathbf{E}^\top$
8: $\quad \left[v_i, v_j\right] \leftarrow \left[v_i - \mu_i, v_j - \mu_j\right] \cdot \mathbf{E}^\top$
9: $\quad \begin{bmatrix} m_{i,i} & m_{i,j} \\ m_{j,i} & m_{j,j} \end{bmatrix} \leftarrow \begin{bmatrix} m_{i,i} & m_{i,j} \\ m_{j,i} & m_{j,j} \end{bmatrix} \cdot \mathbf{E}^\top$
10: **end for**

Trim Box

In this part of the algorithm, the trimming from the box optimization algorithm (see Subsection 2.1.2) is applied to the design points in the rotated coordinate system. However, a few adjustments have to be made before the box can actually be trimmed.

With respect to the new coordinate system determined previously, the solution space Ω_{box} is still a product of one-dimensional intervals I_k and two-dimensional rotated boxes $B_{i,j}$. Nonetheless, the original trimming from [42] is only applicable to axis-parallel boxes. Therefore, Ω_{box} is modified by constructing a bounding box around each rotated box $B_{i,j}$, that is

$$B_{i,j} \subset \left[\widetilde{a}_i, \widetilde{b}_i\right] \times \left[\widetilde{a}_j, \widetilde{b}_j\right],$$

where

$$\left.\begin{aligned} \widetilde{a}_\ell &:= \min\left\{x_\ell \,\middle|\, (x_i, x_j) \in B_{i,j}\right\} \\ \widetilde{b}_\ell &:= \max\left\{x_\ell \,\middle|\, (x_i, x_j) \in B_{i,j}\right\} \end{aligned}\right\} \quad \ell = i, j.$$

This yields a new box that is axis-parallel in the rotated coordinate system (compare Figure 3.5):

$$\widetilde{\Omega}_{\text{box}} := \prod_{i=1}^{d} \left[\widetilde{a}_i, \widetilde{b}_i\right].$$

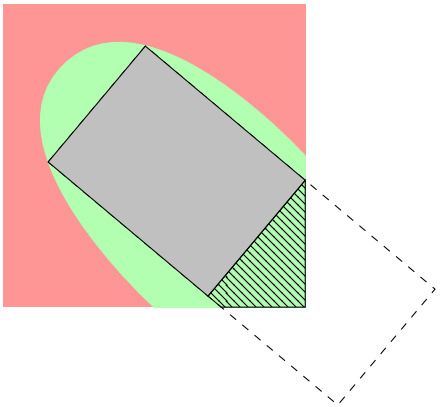

Figure 3.6: Retracting a box. A candidate box outside the design space is adjusted to lie within the design space. The lost design space is hatched.

Now, the original trimming algorithm can be applied to this box, resulting in a box $\Omega^{\star}_{\text{box}}$ that is axis-parallel in the rotated coordinate system. With respect to the original coordinate system, the box has seemingly been rotated such that it lies in those directions of the good design points that maximize their variance.

Grow Box

Compared to the previous steps, not much is changed in this step. However, this step, if applied to rotated boxes, justifies the modifications from Section 2.2. Recall that the box optimization algorithm retracted a box into Ω_{ds} if it had grown outside of Ω_{ds}. While this is a valid practice for axis-parallel boxes, doing the same for a rotated box could induce a potentially significant loss of good design space, as shown in Figure 3.6. Thus, we sample design points in $\Omega_{\text{box}} \cap \Omega_{\text{ds}}$ instead of only Ω_{box}. Also, in each step "Trim Box", when the volume $\mu\big(\Omega^{\star}_{\boldsymbol{x}^{\text{good}}}\big)$ is calculated, the volume $\mu\big(\Omega^{\star}_{\boldsymbol{x}^{\text{good}}} \cap \Omega_{\text{ds,rot}}\big)$ is calculated instead. Here, $\Omega_{\text{ds,rot}}$ is the transformation of $[0,1]^d$ (which is the normalized design space) into the rotated coordinate system prescribed by the 2D-maps,

$$\Omega_{\text{ds,rot}} := \prod_{k \in \mathcal{I}_I} [0,1] \times \prod_{(i,j) \in \mathcal{J}_{\text{pair}}} \rho_{i,j}\left([0,1]^2\right).$$

3.3.3 Consolidation Phase

The consolidation phase from the box optimization algorithm remains unchanged, compare Subsection 2.1.3. Besides no longer being grown, the box is also not rotated any more because its final alignment is considered to be well enough. However, it is still trimmed n^{con} times or until no bad designs are sampled three times in series.

3.4 Numerical Experiments

In this section, we repeat the example problems from Subsection 2.4.1. Afterwards, we study the alignment of the rotated box in the design space.

3.4.1 Two Example Problems in 2D

We apply the rotated box optimization algorithm to the polygon and Rosenbrock problems to get a first impression of the algorithm. Additionally, we compare the results to those of the box optimization algorithm.

2D Polygon

Recall the problem statement from equation (2.4.1). The initial box for the algorithm is chosen as in Subsection 2.4.1,

$$\Omega_{\text{box}} = [1.8, 1.9] \times [2.0, 2.1].$$

We execute the algorithm 100 times, with the same settings as for the box optimization algorithm, i.e. $n^{\text{exp}} = n^{\text{con}} = 100$ steps in the exploration and in the consolidation phase, $N = 100$ design points are sampled in each step, the growth rate is dynamic with $g^{(0)} = 0.05$ and $a^{\text{target}} = 0.6$. The only difference is that we now seek the optimal rotated box with respect to the 2D-map $\Omega_{1,2} := [0, 4] \times [0, 4]$.

The resulting mean normalized volume of the 100 final rotated boxes is 0.1885 with a standard deviation of 0.0174, which is 30% more volume than the box optimization algorithm could find. Figure 3.7 shows again the initial box and a rotated box with normalized volume 0.1887 as well as the optimal axis-parallel box (dashed).

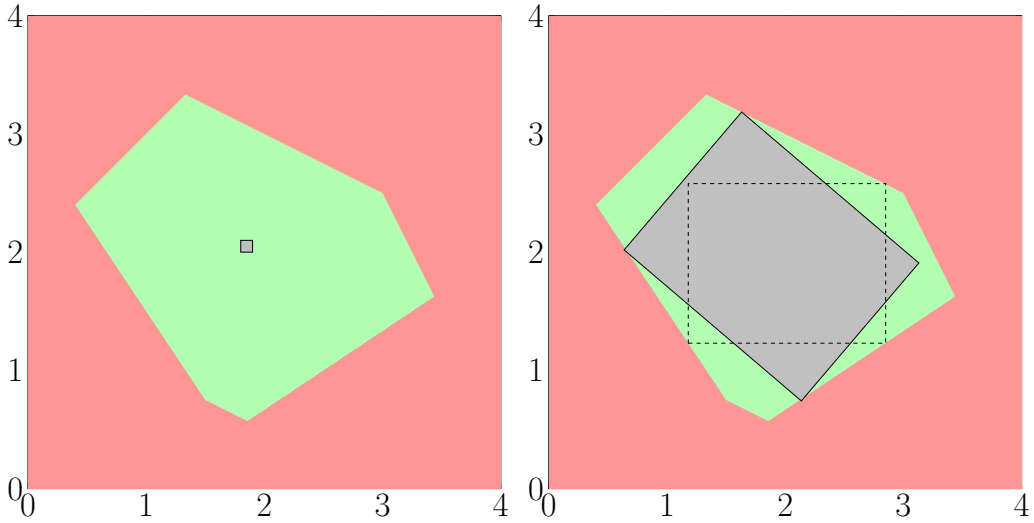

Figure 3.7: Left picture: good design space for the 2D polygon (green) with initial box (gray). Right picture: rotated box with normalized volume 0.1887 (gray) and optimal axis-parallel box (dashed).

Rosenbrock Function

We apply the rotated box algorithm to the Rosenbrock function (see equation (2.4.2)) with the same settings as above. Here, the mean normalized volume of the 100 rotated boxes is 0.0332 with a standard deviation of 0.0015, which is close to the mean normalized volume found by the box optimization.

A rotated box with volume 0.0332 can be seen in Figure 3.8. Note how the rotated box fits tightly into the right part of the U-shape. The algorithm finds more good design space in that region than in the bottom part of the U-shape.

In conclusion, these results suggest that the rotated box optimization algorithm works reasonably well. It seems to find rotated boxes that are at least as large as those found by the box optimization algorithm.

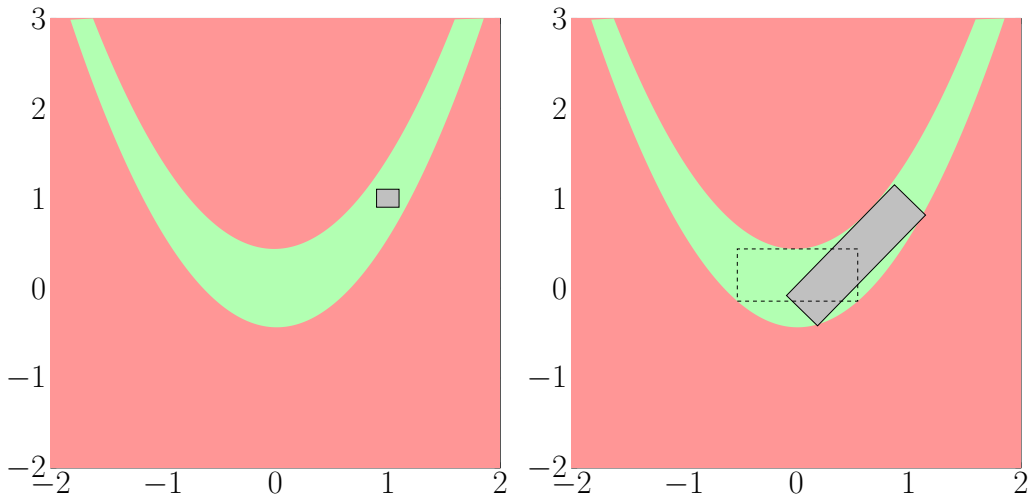

Figure 3.8: Left picture: good design space for the Rosenbrock function with initial box. Right picture: rotated box with normalized volume 0.0332 and optimal axis-parallel box (dashed).

3.4.2 Diagonal Solution Space

It is clear that the principle component analysis converges to the correct angle if the sample size is increased. In order to verify the convergence of the entire rotated box algorithm, we test whether the angles of the rotated boxes converge to the correct angle. To this end, the design space $\Omega_{\text{ds}} := [0,1]^2$ is considered and the objective function is defined such that the good solution space generates a diagonal corridor of width $\frac{1}{\sqrt{2}}$ in the design space. The angle between the corridor and the x-axis is 45°, see Figure 3.9 for an illustration. The growth of the box is dynamic, with $a^{\text{target}} = 0.8$ and an initial growth rate of 0.1.

We repeat the algorithm 100 times each for the sample sizes $N = 50, 60, 70, \ldots, 500$. For each final box, the angle between the box and the x-axis is calculated. The mean of the angles is close to 45° for all sample sizes, and the standard deviation of 2.5° does not change significantly for sample sizes larger than about 200. The

observed results form a funnel, compare Figure 3.10. This can be interpreted as a sign that, with growing sample sizes, the angle of the final rotated box converges to 45°.

Next, the rotated box algorithm is tested for the corridor widths $0.05, 0.1, \ldots, 0.75$, with the number of design points fixed to 100. The algorithm is executed 100 times for each width, skipping the consolidation phase in every execution.

As can be seen in Figure 3.11, the angle of the box varies the more the wider the corridors become. This is due to the fact that, for a large corridor size, the box moves into the region outside the design space, where no design points are sampled. Thus, the final position of the box varies a lot more. This observation is confirmed by Figure 3.12, where the solution boxes are found for the corridor width 0.1 (left picture) and the corridor width 0.75 (right picture). The non-defined space contained in the solution box is much larger in the right picture than in the left picture. However, the mean stays very close to 45° throughout the whole test, and the standard deviation is only about 5° for a corridor of width 0.75, compare Figure 3.11. This suggests that the resulting boxes again converge to 45° and the algorithm works as intended.

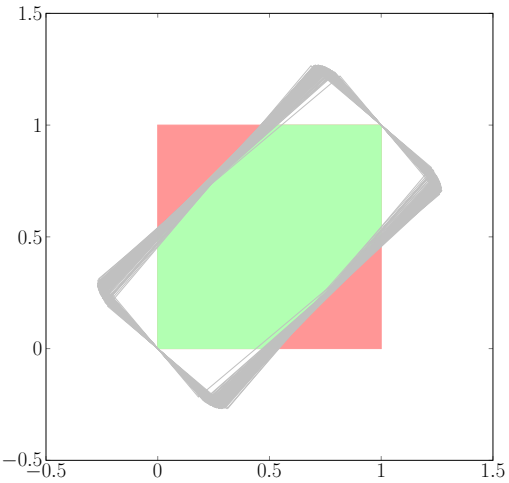

Figure 3.9: 100 solution spaces for the sample size $N = 500$ for the diagonal solution space problem.

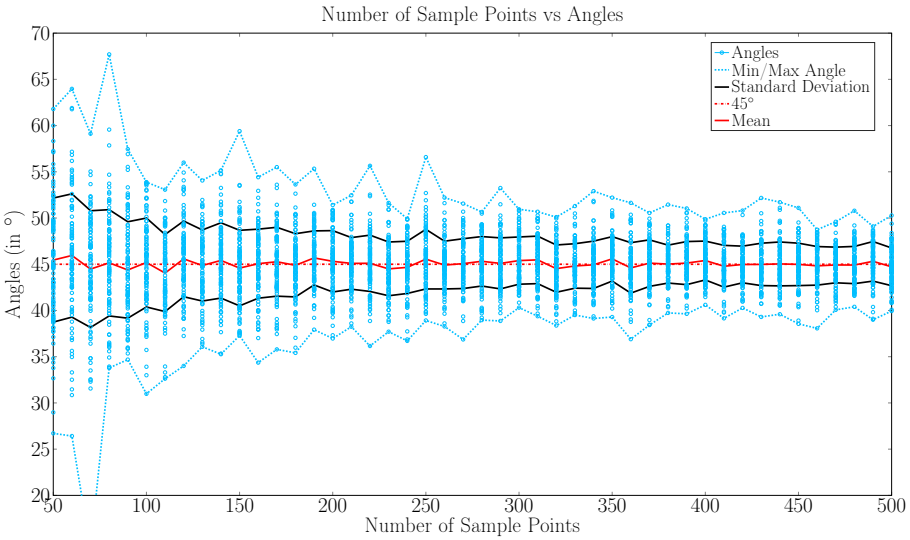

Figure 3.10: Funnel for the sample sizes from 50 to 500.

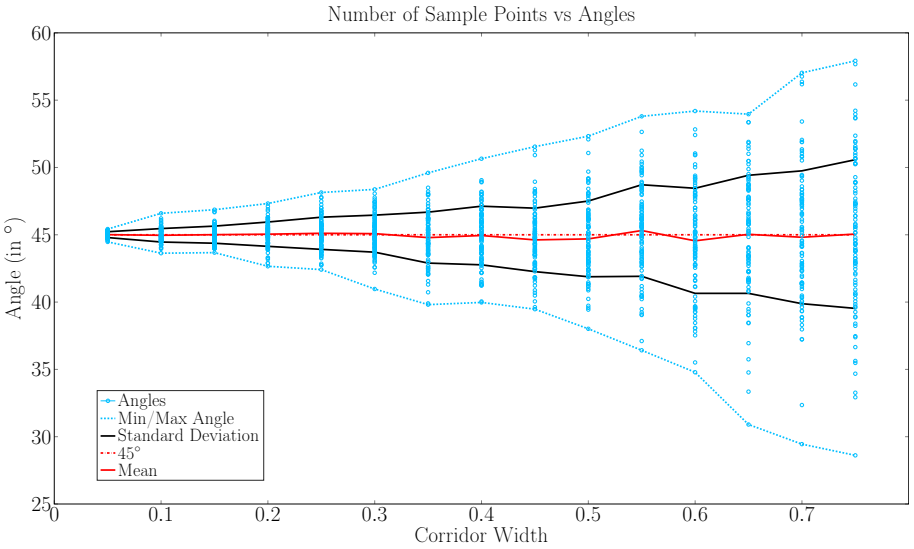

Figure 3.11: The test results for varying corridor widths.

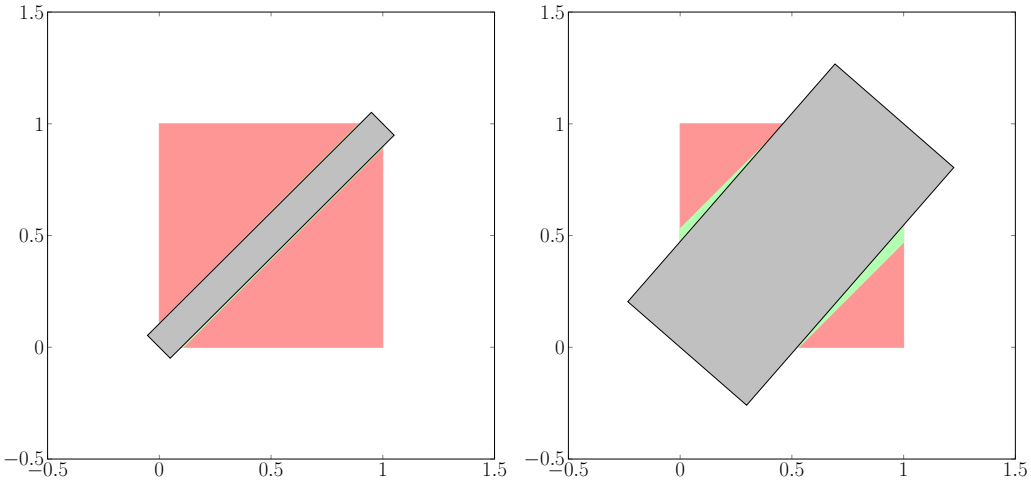

Figure 3.12: The final box for a corridor of width 0.1 (left) and for a corridor of width 0.75 (right).

Chapter 4

Polytope Optimization

In this chapter, we present an algorithm that extends the idea of the rotated box algorithm from Chapter 3. Because a coupling of variables is already introduced by considering rotated boxes, we try to take even more advantage of this. We wish to find a type of solution space that generally has a larger volume in a given good design space than a rotated box. A choice that quickly comes to mind is a polygon. As illustrated in Figure 4.1 for the Rosenbrock function, replacing a box with a polygon could yield significantly more volume. Thus, we allow pairs of design variables on a 2D-map to lie within polygon-shaped solution spaces. This means that the solution space is a product of two-dimensional polygons. Since the equivalent of polygons in higher dimensions are called *polytopes*, the algorithm is referred to as *polytope optimization algorithm*.

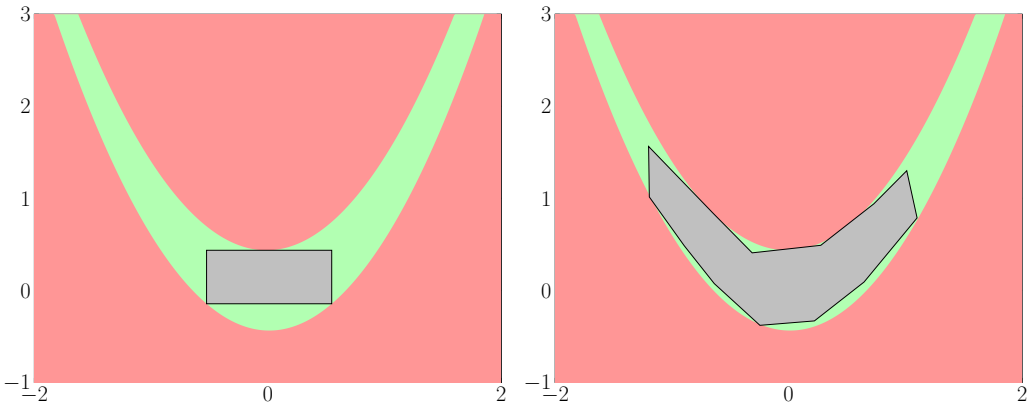

Figure 4.1: An axis-parallel solution space (left) and a polygon-shaped solution space (right) inside the U-shaped good design space of the Rosenbrock function.

4.1 Polytopes for 2D-Maps

As mentioned above, the polytope optimization algorithm is conceptually identical to the rotated box optimization algorithm. However, the axis-parallel hyperbox Ω_{box} is replaced by a solution space Ω_{pol} which is a product of one-dimensional intervals

I_k and two-dimensional polygons $P_{i,j}$,

$$\Omega_{\text{pol}} := \prod_{k \in \mathcal{J}_{\text{int}}} I_k \times \prod_{(i,j) \in \mathcal{J}_{\text{pair}}} P_{i,j}, \tag{4.1.1}$$

where $\mathcal{J}_P := \{1, \dots, d\} \setminus \mathcal{J}_{\text{int}}$ and

$$\mathcal{J}_{\text{int}} := \big\{ i \in \{1, \dots, d\} \,\big|\, i \text{ is an unpaired dimension} \big\},$$
$$\mathcal{J}_{\text{pair}} := \big\{ (i,j) \in \mathcal{J}_P \times \mathcal{J}_P \,\big|\, \text{the dimensions } i \text{ and } j \text{ are coupled} \big\}.$$

Note that design variables that have not to be paired with another variable are assigned to the intervals I_k. The solution space Ω_{pol} is thus a specific high-dimensional polytope. As an example, Figure 4.2 depicts a solution space Ω_{pol} in three dimensions. If Ω_{pol} is a product of polygons only, i.e. if

$$\Omega_{\text{pol}} = \prod_{(i,j) \in \mathcal{J}_{\text{pair}}} P_{i,j},$$

it is a *product prism*, which is the term for a polytope that is a product of polytopes with two or more dimensions (see [17]). Likewise, we call a polytope that can be written as in (4.1.1) a *product polytope*. We define the space of *admissible* product polytopes as

$$\mathcal{S}_{\text{prod}} := \left\{ \Omega_{\text{pol}} \subset \Omega_{\text{ds}} \,\middle|\, \Omega_{\text{pol}} = \prod_{k \in \mathcal{J}_{\text{int}}} I_k \times \prod_{(i,j) \in \mathcal{J}_{\text{pair}}} P_{i,j} \right\}.$$

Finally, problem (1.4.1) can be rewritten in the following way:

$$\begin{cases} \text{Maximize the volume } \mu\left(\Omega_{\text{pol}}\right) \\ \text{over all product polytopes } \Omega_{\text{pol}} \in \mathcal{S}_{\text{prod}} \\ \text{subject to } f(\boldsymbol{x}) \leq c \text{ for all } \boldsymbol{x} \in \Omega_{\text{pol}}. \end{cases} \tag{4.1.2}$$

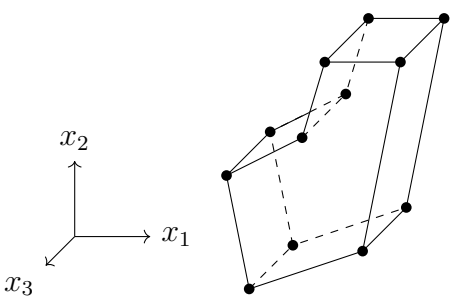

Figure 4.2: Visualization of a polytope $\Omega_{\text{pol}} := I_3 \times P_{1,2}$ in three dimensions, where $I_3 := [0, 1.5]$ and $P_{i,j} := \big\{(0,-1), (1.5,-0.5), (2,2), (1,2), (0.7,1), (-0.3,0.5)\big\}$.

4.2 Manipulating 2D Polygons

This section intends to give an overview of the elementary manipulation steps of two-dimensional polygons. They are the the basic ingredients for the polytope optimization algorithm presented in Section 4.3. A particular polygon P has a fixed number M of vertices and is represented by the ordered sequence of these vertices, that is

$$P := \left\{ \boldsymbol{v}^{(1)}, \dots, \boldsymbol{v}^{(M)} \right\}.$$

4.2.1 Sample Design Points

A design point inside the polygon P is obtained by constructing a bounding box around P and sampling uniformly distributed random points inside the bounding box until a point $\boldsymbol{x} \in \mathbb{R}^2$ is found that also lies within P. Determining whether a point lies within a polygon can be done via the winding number algorithm (see [47] and Scetion 4.2.2). After a fixed number of design points have been sampled, all design points $\boldsymbol{x}$ are evaluated with the objective function f and then marked as good or bad points, see Figure 4.3.

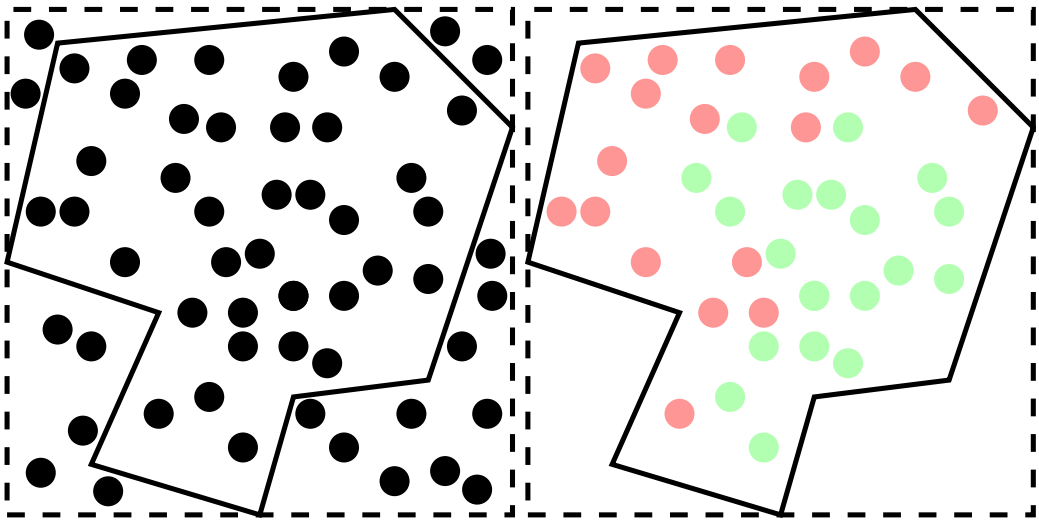

Figure 4.3: Points are sampled in the dashed bounding box (left), then design points outside the box are removed and the remaining ones are marked as good or bad (right).

4.2.2 Winding Number Algorithm

The winding number algorithm is the centerpiece of the sampling step. It allows us to efficiently find points inside a polygon.

Definition 4.2.1. *Let $\boldsymbol{\gamma}$ be a closed curve in $\mathbb{R}^2$, that is, $\boldsymbol{\gamma}(t) := \big(\gamma_1(t), \gamma_2(t)\big)^\top$ for $t \in [a, b]$, where $\boldsymbol{\gamma}(a) = \boldsymbol{\gamma}(b)$. For $\boldsymbol{x} \in \mathbb{R}^2 \setminus \boldsymbol{\gamma}\big([a, b]\big)$, the* winding number *$\omega(\boldsymbol{x}, \boldsymbol{\gamma}) \in \mathbb{Z}$ is the number of counterclockwise circulations made around $\boldsymbol{x}$ while moving along $\boldsymbol{\gamma}$.*

A point $\boldsymbol{x}$ with $\omega(\boldsymbol{x}, \boldsymbol{\gamma}) = 0$ lies outside the curve $\boldsymbol{\gamma}$, while a point with $\omega(\boldsymbol{x}, \boldsymbol{\gamma}) = 1$ lies inside $\boldsymbol{\gamma}$. If $\boldsymbol{\gamma}$ is a curve with one or more self-intersections, $\omega(\boldsymbol{x}, \boldsymbol{\gamma})$ may admit other values in $\mathbb{Z}$, see Figure 4.4 for a particular example. We say that all points $\boldsymbol{x}$ with an even winding number $\omega(\boldsymbol{x}, \boldsymbol{\gamma})$ lie *outside* of $\boldsymbol{\gamma}$ and all points $\boldsymbol{x}$ with an odd winding number $\omega(\boldsymbol{x}, \boldsymbol{\gamma})$ lie *inside* of $\boldsymbol{\gamma}$.

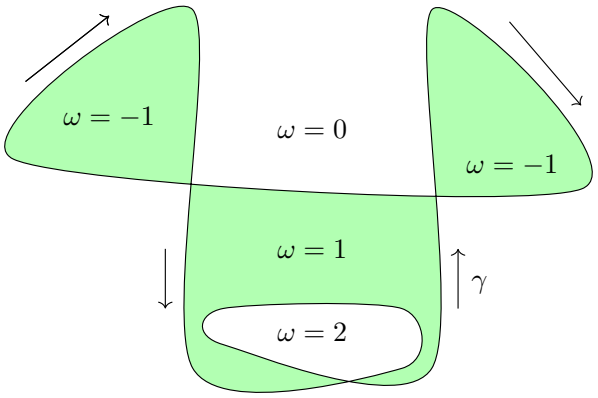

Figure 4.4: Regions inside a curve $\boldsymbol{\gamma}$ with associated winding numbers ω.

The following winding number algorithm is presented in [47] and [74]:

For a given $\boldsymbol{x}$ and t, consider the line through $\boldsymbol{x}$ and $\boldsymbol{\gamma}(t)$ and the line through $\boldsymbol{x}$ that is parallel to the x-axis. Define $\phi(t)$ as the angle between those two lines (compare the left picture of Figure 4.5). We can assume, without loss of generality, that $\boldsymbol{x} = (0, 0)^\top$ is the origin. Then, $\phi(t) = \arctan\big(\gamma_1(t)/\gamma_2(t)\big)$ and it follows that

$$\omega(\boldsymbol{x}, \boldsymbol{\gamma}) := \frac{1}{2\pi} \int_{\phi(a)}^{\phi(b)} \mathrm{d}\phi = \frac{1}{2\pi} \int_a^b \frac{\mathrm{d}\phi}{\mathrm{d}t}(t)\, \mathrm{d}t = \frac{1}{2\pi} \int_a^b \frac{\gamma_2'(t)\gamma_1(t) - \gamma_2(t)\gamma_1'(t)}{\gamma_1(t)^2 + \gamma_2(t)^2}\, \mathrm{d}t.$$

A polygon $P = \big\{\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(M)}\big\}$ can be interpreted as a piecewise linear curve $\boldsymbol{\rho} : [1, M+1] \to \mathbb{R}^2$ with

$$\boldsymbol{\rho}(s) := \big(x_i(s-i), y_i(s-i)\big)^\top$$

for $s \in [i, i+1]$, $i = 1, \ldots, M$, and

$$\big(x_i(t), y_i(t)\big)^\top := \begin{cases} t\boldsymbol{v}^{(i+1)} + (1-t)\boldsymbol{v}^{(i)}, & i = 1, \ldots, M-1, \\ t\boldsymbol{v}^{(1)} + (1-t)\boldsymbol{v}^{(M)}, & i = M, \end{cases}$$

for $t \in [0, 1]$. The winding number for a point inside a polygon can then be calculated

by

$$\begin{aligned}\omega(\boldsymbol{x},\boldsymbol{\rho}) &= \frac{1}{2\pi}\sum_{i=1}^{M}\int_0^1 \frac{y_i'(t)x_i(t)-y_i(t)x_i'(t)}{x_i(t)^2+y_i(t)^2}\,\mathrm{d}t \\ &= \frac{1}{2\pi}\sum_{i=1}^{M-1}\arccos\left(\frac{\boldsymbol{v}^{(i)}\times\boldsymbol{v}^{(i+1)}}{\|\boldsymbol{v}^{(i)}\|\|\boldsymbol{v}^{(i+1)}\|}\right)\cdot\operatorname{sign}\begin{vmatrix} v_1^{(i)} & v_1^{(i+1)} \\ v_2^{(i)} & v_2^{(i+1)}\end{vmatrix} \\ &+ \frac{1}{2\pi}\arccos\left(\frac{\boldsymbol{v}^{(1)}\times\boldsymbol{v}^{(M)}}{\|\boldsymbol{v}^{(1)}\|\|\boldsymbol{v}^{(M)}\|}\right)\cdot\operatorname{sign}\begin{vmatrix} v_1^{(1)} & v_1^{(M)} \\ v_2^{(1)} & v_2^{(M)}\end{vmatrix} \\ &= \frac{1}{2\pi}\sum_{i=1}^{M-1}\phi_i,\end{aligned}$$

where ϕ_i is the signed angle between the line from $\boldsymbol{x}$ to $\boldsymbol{v}^{(i)}$ and the line from $\boldsymbol{x}$ to $\boldsymbol{v}^{(i+1)}$ (compare the right picture of Figure 4.5).

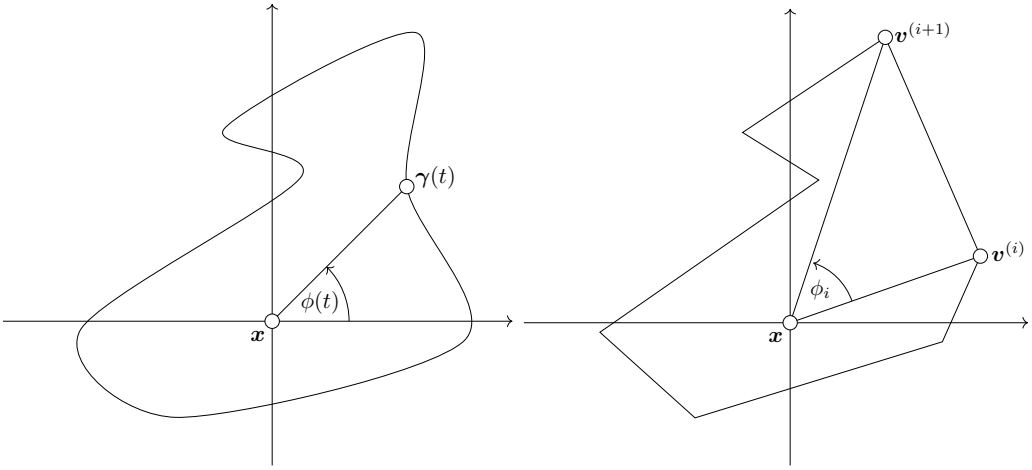

Figure 4.5: Illustration of the angles $\phi(t)$ and ϕ_i.

However, calculating the winding number with this formula requires the use of the computationally expensive arccosine function. Such computations can be avoided with the aid of the following observation.

Define $\boldsymbol{e}_i$ as the edge from $\boldsymbol{v}^{(i)}$ to $\boldsymbol{v}^{(i+1)}$. If we emit a ray from $\boldsymbol{x}$, the winding number can be calculated by counting how many edges $\boldsymbol{e}_i$ cross the ray. If $\boldsymbol{e}_i$ crosses the ray from below to above, i.e. $\boldsymbol{v}^{(i)}$ lies below the ray and $\boldsymbol{v}^{(i+1)}$ above, we add 1 to the winding number. On the other hand, if $\boldsymbol{e}_i$ crosses the ray from above to below, i.e. $\boldsymbol{v}^{(i)}$ lies above the ray and $\boldsymbol{v}^{(i+1)}$ below, we subtract 1 from the winding number. We demonstrate this in Figure 4.6, where rays are emitted from multiple points and the winding number of a point is adjusted whenever an edge crosses its associated ray.

Note that it is not necessary to calculate intersections of the ray and the polygon's edges. We can tell whether the ray crosses an edge by undertaking a few geometrical considerations.

First, for determining the winding number, it does not matter in which direction the ray shows. Thus, we always choose the ray to be parallel to the horizontal axis.

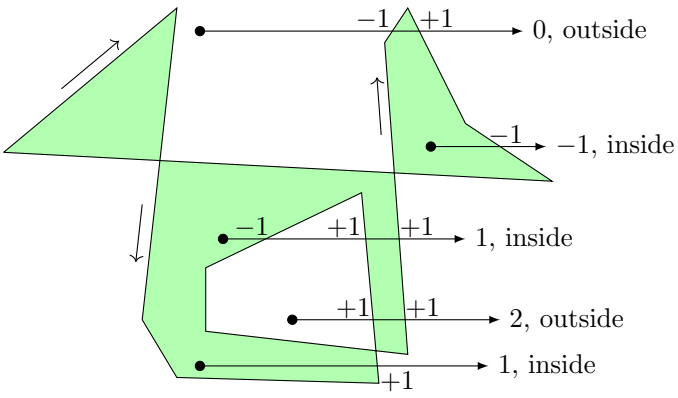

Figure 4.6: Calculating the winding number for points in relation to a polygon.

Second, this horizontal ray will be crossed by an edge $\boldsymbol{e}_i$ if and only if the vertical coordinate of $\boldsymbol{x}$ lies between the vertical coordinates of $\boldsymbol{v}^{(i)}$ and $\boldsymbol{v}^{(i+1)}$, i.e., if either $v_2^{(i)} \leq x_2 < v_2^{(i+1)}$ for an edge crossing from below to above or $v_2^{(i+1)} \leq x_2 < v_2^{(i)}$ for an edge crossing from above to below. If none of these inequalities are true, there is no change in the winding number.

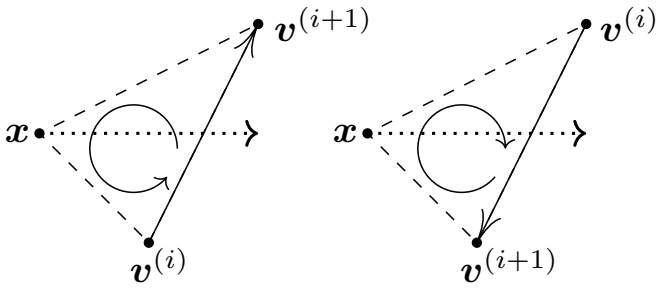

Figure 4.7: Orientation of the triangle $\Delta\big(\boldsymbol{v}^{(i)}, \boldsymbol{v}^{(i+1)}, \boldsymbol{x}\big)$.

Third, from the previous two observations it follows that we only need to determine whether the ray is crossed by the edge $\boldsymbol{e}_i$ from below to above or vice versa. To this end, we consider the triangle $\Delta\big(\boldsymbol{v}^{(i)}, \boldsymbol{v}^{(i+1)}, \boldsymbol{x}\big)$. If this triangle is orientated counterclockwise, i.e. if its *signed area*

$$A_i := \frac{1}{2}\left(\boldsymbol{v}^{(i+1)} - \boldsymbol{v}^{(i)}\right) \times \left(\boldsymbol{x} - \boldsymbol{v}^{(i)}\right)$$

fulfills $A_i > 0$, $\boldsymbol{e}_i$ crosses the ray from below to above (see the left picture of Figure 4.7). Otherwise, if $A_i < 0$, then it is orientated clockwise and $\boldsymbol{e}_i$ crosses the ray from above to below (see the right picture of Figure 4.7).

From these observations, we can formulate an algorithm that determines the winding number of a point with respect to a polygon, see Algorithm 4. It allows us to determine the points with an odd winding number, which are per definition those that lie within a given polygon.

Algorithm 4 (Winding Number). This algorithm calculates the winding number of a point $\boldsymbol{x}$ with respect to a polygon P.

Input: $\boldsymbol{x}, P$
Output: ω

$\omega \leftarrow 0$
for all edges $\boldsymbol{e}_i$ **do**
    if $v_2^{(i)} \leq x_2 < v_2^{(i+1)}$ **and** $A_i > 0$ **then**
        $\omega \leftarrow \omega + 1$
    else if $v_2^{(i+1)} \leq x_2 < v_2^{(i)}$ **and** $A_i < 0$ **then**
        $\omega \leftarrow \omega - 1$
    end if
end for

4.2.3 Trim Polygons

In order to find the good design space, a polygon needs to be trimmed such that it contains no bad sample points. This is done by successively removing bad sample points from the polygon. To accomplish this, a bad sample point is specified. A good sample point is chosen (see left picture in Figure 4.8) and a triangle out of this good sample point and two neighboring vertices is formed, such that the bad sample point is contained in this triangle (see right picture in Figure 4.8). From this constellation, there exist multiple strategies how we can remove the bad point. We present the three most promising procedures in the following:

- **Edge Walking Strategy.** With the "Edge Walking" strategy, each vertex of the two polygon vertices "walks" along the edges of the triangle until the boundary of the polygon lies on the bad sample point, i.e. the vertices are moved towards the good sample point on the edges of the triangle until the bad sample point lies on the edge of the polygon (see Figure 4.9). Thus, the bad sample point no longer lies in the polygon. This strategy yields two possible polygons, one for each vertex. We will decide later which of these polygons we keep.
- **Move Vertex Strategy.** By applying the "Move Vertex" strategy, we simply move each of the two vertices directly onto the bad point, removing it from the polygon (see Figure 4.10). This strategy also yields a polygon for each vertex that is moved, only one of which is kept.
- **Move Edge Strategy.** The "Move Edge" strategy takes the edge of the triangle that lies on the polygon's boundary and moves that edge until it lies on the bad point. The edge is thereby kept parallel to its former position (see Figure 4.11). Only one polygon results from this strategy.

One of these strategies is chosen and then repeated for each good sample point, yielding multiple polygons that are differently trimmed. From those, the best polygon (according to the quality measures which we introduce in Subsection 4.2.4) is chosen as the new, trimmed polygon. Then, this procedure is repeated again for all bad sample points remaining in the trimmed polygon.

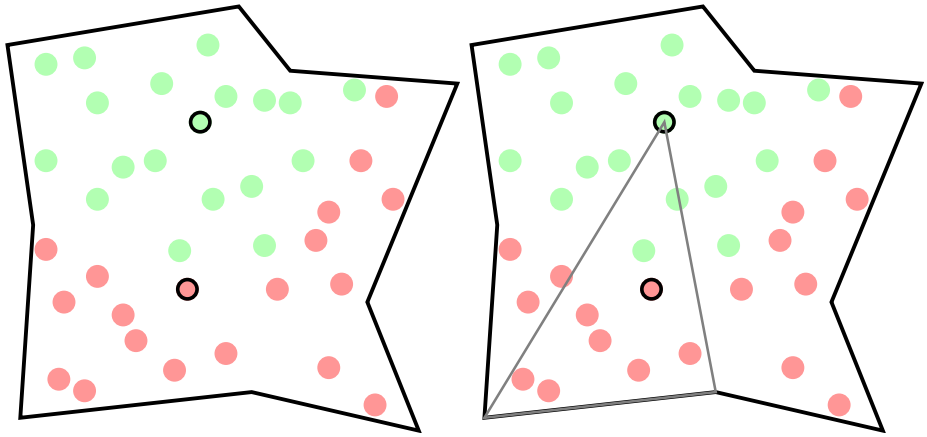

Figure 4.8: From a good and a bad point (left) a triangle is constructed (right).

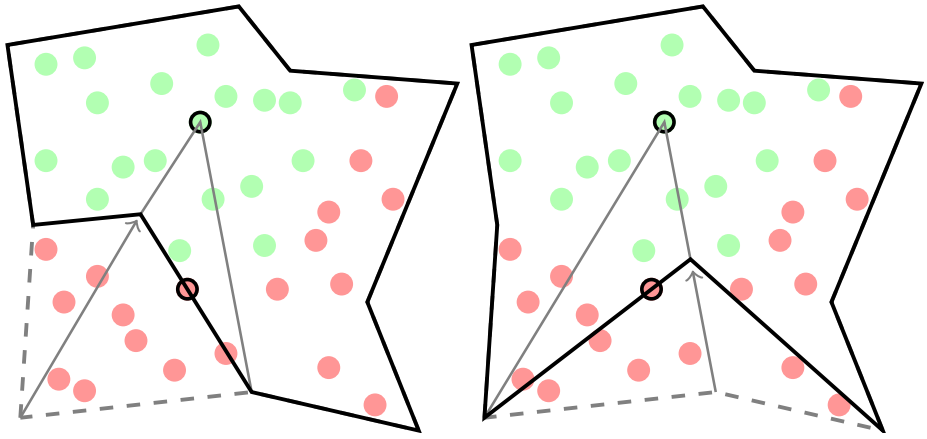

Figure 4.9: Edge Walking: The two vertices are moved toward the good point for two possible polygons (left and right).

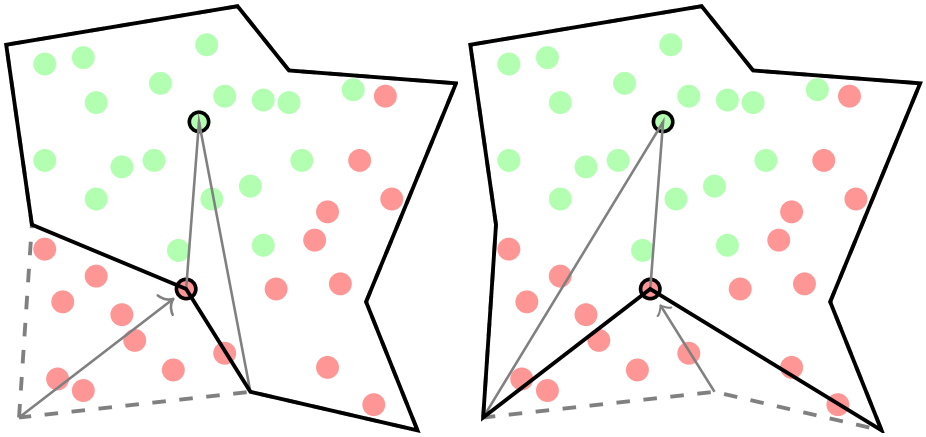

Figure 4.10: Move Vertex: Each vertex is move onto the bad point for two possible polygons (left and right).

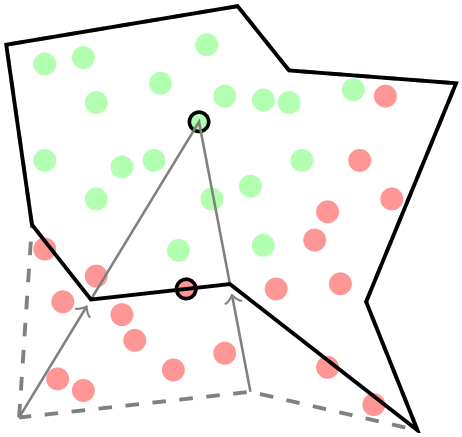

Figure 4.11: Move Edge: The edge of the triangle is moved until it lies on the bad point, staying parallel to its former position.

Algorithms

The details of the polygon trimming can be found in Algorithm 5. It requires a polygon P with vertices $\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(M)}$, a good point $\boldsymbol{x}^{\text{good}}$ and a bad point $\boldsymbol{x}^{\text{bad}}$ as inputs (line 1). For each vertex $\boldsymbol{v}^{(k)}$, it is checked whether the bad point lies within the convex hull of $\boldsymbol{x}^{\text{good}}$, $\boldsymbol{v}^{(k)}$ and $\boldsymbol{v}^{(k+1)}$, which is exactly the triangle formed by those points (lines 3 and 4). If $\boldsymbol{x}^{\text{bad}}$ does lie within the triangle, the polygon is trimmed as explained above by Algorithm 6 (lines 5 and 6). The two possible outcomes are evaluated with the quality measures from Subsection 4.2.4 and the better one is kept (line 7). Note that we skip lines 6 and 7 if we pursue the strategy "Move Edge", as this strategy only yields one possible outcome. Finally, the best polygon $P^{(k)}$ is chosen as output in line 10.

Algorithm 5 (Trim Polygon). Trim a polygon, keeping as many good sample points as possible.

1: **Input:** P, $\boldsymbol{x}^{\text{good}}$, $\boldsymbol{x}^{\text{bad}}$
2: **Output:** P

3: **for all** $\boldsymbol{v}^{(k)} \in P$ **do**
4: **if** $\boldsymbol{x}^{\text{bad}} \in \text{conv}\left(\left\{\boldsymbol{x}^{\text{good}}, \boldsymbol{v}^{(k)}, \boldsymbol{v}^{(k+1)}\right\}\right)$ **then**
5: $P_1 \leftarrow \texttt{trim\_triangle}\left(P, \boldsymbol{x}^{\text{good}}, \boldsymbol{x}^{\text{bad}}, \boldsymbol{v}^{(k)}, \boldsymbol{v}^{(k+1)}\right)$
6: $P_2 \leftarrow \texttt{trim\_triangle}\left(P, \boldsymbol{x}^{\text{good}}, \boldsymbol{x}^{\text{bad}}, \boldsymbol{v}^{(k+1)}, \boldsymbol{v}^{(k)}\right)$
7: $P^{(k)} \leftarrow \texttt{evaluate}(P_1, P_2)$
8: **end if**
9: **end for**
10: $P \leftarrow \texttt{evaluate}\left(\left\{P^{(k)}\right\}_{k=1}^{N}\right)$

The procedure `trim_triangle` is defined in Algorithm 6. It takes a polygon P, a good point $\boldsymbol{x}^{\text{good}}$, a bad point $\boldsymbol{x}^{\text{bad}}$, two neighboring vertices $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ and a trimming strategy as input arguments (line 1).

If we follow the strategy "Edge Walking", the triangle formed by $\boldsymbol{x}^{\text{good}}$, $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ is trimmed by moving the edge between $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ such that it lies on $\boldsymbol{x}^{\text{bad}}$. The edges of the triangle are initialized in lines 4 and 5. Then, in line 6, the linear system of equations

$$[\mathbf{e}_1, -\mathbf{e}_2] \cdot \mathbf{t} = \boldsymbol{v}_2 - \boldsymbol{v}_1$$

is solved and the value t_1 is used to determine how far the vertex $\boldsymbol{v}_1$ has to be moved (line 7). Finally, the corresponding vertex in the polygon is updated (line 8).

If the strategy is "Move Vertex", $\boldsymbol{v}_1$ is simply replaced with $\boldsymbol{x}^{\text{bad}}$ (lines 10 and 11).

Finally, for the strategy "Move Edge", we initialize all three edges of the triangle in lines 13 to 15. Then, we solve the linear system of equations

$$[\mathbf{d}, -\mathbf{e}_1] \cdot \mathbf{a} = \boldsymbol{x}^{\text{good}} - \boldsymbol{x}^{\text{bad}}$$

and

$$[\mathbf{d}, -\mathbf{e}_2] \cdot \mathbf{b} = \boldsymbol{x}^{\text{good}} - \boldsymbol{x}^{\text{bad}}$$

in lines 16 and 17. The values a_1 and b_1 tell us how far the vertices $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ have to be moved along the edges of the triangle such that the edge of the polygon stays parallel to its former position and lies on $\boldsymbol{x}^{\text{bad}}$ (lines 18 and 19). Finally, we update the polygon in lines 20 and 21.

Algorithm 6 (Trim Triangle). Trim a triangle inside a polygon.

1: **Input:** P, $\boldsymbol{x}^{\text{good}}$, $\boldsymbol{x}^{\text{bad}}$, $\boldsymbol{v}_1$, $\boldsymbol{v}_2$, strategy
2: **Output:** P

3: **if** strategy = "Edge Walking" **then**
4: $\quad \mathbf{e}_1 \leftarrow \boldsymbol{x}^{\text{good}} - \boldsymbol{v}_1$
5: $\quad \mathbf{e}_2 \leftarrow \boldsymbol{x}^{\text{bad}} - \boldsymbol{v}_2$
6: $\quad$ Solve $[\mathbf{e}_1, -\mathbf{e}_2] \cdot \mathbf{t} = \boldsymbol{v}_2 - \boldsymbol{v}_1$
7: $\quad \boldsymbol{v}_1 \leftarrow \boldsymbol{v}_1 + t_1 \cdot \mathbf{e}_1$
8: $\quad P \leftarrow \texttt{update}(P, \boldsymbol{v}_1)$
9: **else if** strategy = "Move Vertex" **then**
10: $\quad \boldsymbol{v}_1 \leftarrow \boldsymbol{x}^{\text{bad}}$
11: $\quad P \leftarrow \texttt{update}(P, \boldsymbol{v}_1)$
12: **else if** strategy = "Move Edge" **then**
13: $\quad \mathbf{e}_1 \leftarrow \boldsymbol{v}_1 - \boldsymbol{x}^{\text{good}}$
14: $\quad \mathbf{e}_2 \leftarrow \boldsymbol{v}_2 - \boldsymbol{x}^{\text{good}}$
15: $\quad \mathbf{d} \leftarrow \boldsymbol{v}_2 - \boldsymbol{v}_1$
16: $\quad$ Solve $[\mathbf{d}, -\mathbf{e}_1] \cdot \mathbf{a} = \boldsymbol{x}^{\text{good}} - \boldsymbol{x}^{\text{bad}}$
17: $\quad$ Solve $[\mathbf{d}, -\mathbf{e}_2] \cdot \mathbf{b} = \boldsymbol{x}^{\text{good}} - \boldsymbol{x}^{\text{bad}}$
18: $\quad \boldsymbol{v}_1 \leftarrow \boldsymbol{x}^{\text{good}} + a_1 \cdot \mathbf{e}_1$
19: $\quad \boldsymbol{v}_2 \leftarrow \boldsymbol{x}^{\text{good}} + b_1 \cdot \mathbf{e}_2$
20: $\quad P \leftarrow \texttt{update}(P, \boldsymbol{v}_1)$
21: $\quad P \leftarrow \texttt{update}(P, \boldsymbol{v}_2)$
22: **end if**

4.2.4 Evaluation of Polygons

As multiple trimmed polygons are obtained at several steps of the optimization algorithm, the best polygon has to be chosen from among those polygons. For this purpose, quality measures for the polygons need to be introduced. A polygon not fulfilling one of these measures is immediately rejected and not used further in the algorithm. The polygons are rated as follows:

- **Minimum Number of Self-Intersections.** Each polygon should be free of self-intersections. Self-intersections lead to unwanted behaviour of the algorithms. It is not clear how to trim a polygon with self-intersections, and multiple self-intersections overlaying each other obscure what the interior of the polygon is. Thus, polygons having no self-intersections are preferred over polygons with self-intersections (see Figure 4.12).

- **Minimum/Maximum Size of Angles.** Polygons with very small angles or very large angles form spikes, see Figure 4.13. When a spike is trimmed, it

is very likely that a self-intersection is induced. Additionally, there is only a small chance for a point to be sampled within a spike, which in turn means that the spike will not be removed in a trimming step, making the vertex in the corner of the spike redundant. For these reasons, polytopes with no or only a few spikes are preferred. For a fixed threshold angle α, polygons which satisfy $\alpha < \phi < 2\pi - \alpha$ for as many vertex angles ϕ as possible (see Figure 4.13) are favored over others.

- **Maximum Number of Good Points.** Finally, after rejecting all polygons with a bad shape, the size of the good design space is considered. Therefore, the numbers of good points within the polygons are compared and the polygon containing the most is chosen. If that polygon is not unique, i.e. because multiple polygons contain the same highest number of points, one among them is chosen at random (see Figure 4.14).

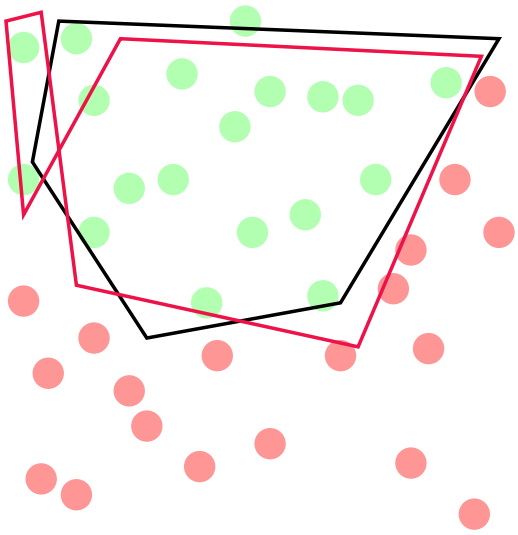

Figure 4.12: The black polygon is preferred over the red polygon because it has no self-intersections.

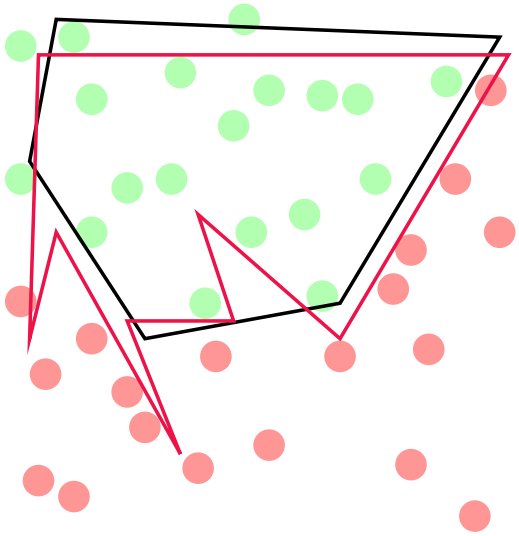

Figure 4.13: The black polygon is preferred over the red polygon because it contains fewer spikes.

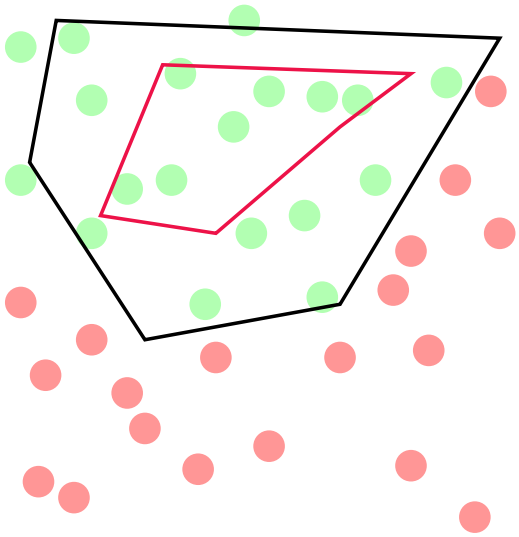

Figure 4.14: The black polygon is preferred over the red polygon because it has more good points.

4.2.5 Remove Spikes

After trimming and evaluating the polygon, it might still contain spikes. If this is the case, i.e. if there are vertices whose angles ϕ violate the condition $\alpha < \phi < 2\pi - \alpha$, then these vertices are relocated (compare Figure 4.15). After this step, all spikes of the polygon are removed. The polygon does not lose much volume by this operation, as the spikes only have a very small volume.

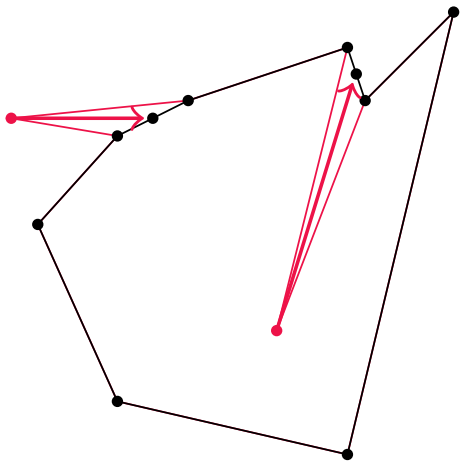

Figure 4.15: Relocating vertices in order to remove spikes (red) from a polygon.

4.2.6 Relocate Vertices

A further manipulation step consists of relocating vertices. The idea behind this is to avoid a degeneration of the polygon, and especially prevent vertices from clustering in one part of the polygon. This reduces the risk of a polygon developing new spikes.

The strategy of the relocation is as follows: The shortest edge of the polygon is removed by replacing its endpoints by the midpoint of the edge. Hence, one vertex is removed from the polygon. To keep the number of vertices constant, a new vertex is placed at the midpoint of the longest edge of the polygon (see Figure 4.16).

4.2.7 Grow Polygon

In this step, the polygon is grown in all directions. This allows the polygon to extend into regions of good design space. Every vertex of the polygon is moved by the same factor g along its outward pointing angle bisector (see Figure 4.17). The vector of the angle bisector is normalized to 1.

4.2.8 Retract Polygon

After a polygon has grown out of the design space Ω_{ds} in the growth step, we retract it into Ω_{ds}. Because we only sample designs within Ω_{ds}, vertices that are outside of it might never be trimmed such that they return into Ω_{ds}. Instead, they are moved

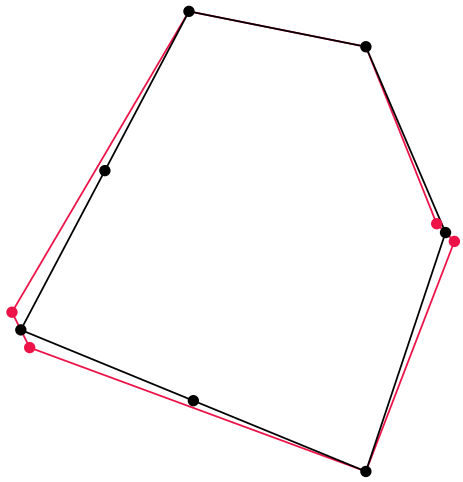

Figure 4.16: Two short edges (red) are removed from a polygon, then two vertices are added on the longest edges.

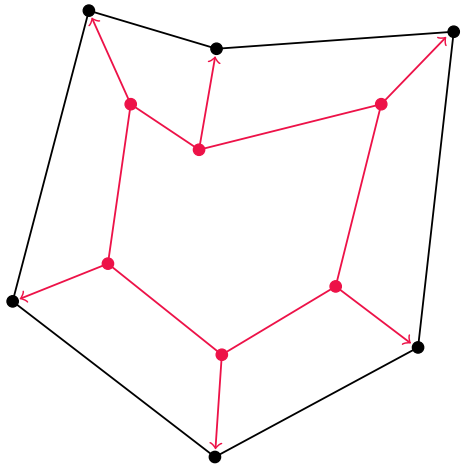

Figure 4.17: Growing a polygon.

further away from Ω_{ds} with each growth step. This leads to a polygon that becomes arbitrarily large and mostly lies outside of Ω_{ds}. Because it is very unlikely that good designs are sampled in such a polygon, we have to perform a retraction step after each growth step.

In the retraction step, we move each vertex outside of Ω_{ds} along the inward pointing angle bisectors of the previously grown polygon until it lies on the boundary of Ω_{ds}. An example of this procedure can be seen in Figure 4.18. After the polygon (red) is grown (dotted), two of its vertices lie outside of Ω_{ds} (dashed). They are moved along their angle bisectors until they lie on the boundary of Ω_{ds}, forming the retracted polygon (black).

4.2.9 Remove Self-Intersections

Sometimes, self-intersections get introduced to the polygon through the trimming, growing, and relocating steps, despite the quality measures trying to prevent this.

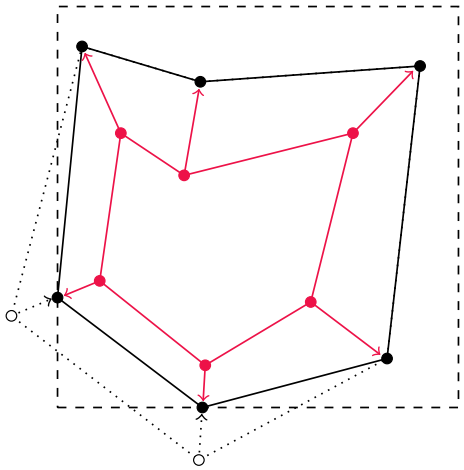

Figure 4.18: Retracting a polygon along the inward pointing angle bisectors of the previously grown polygon (dotted).

Therefore, an algorithm is presented that removes those self-intersections.

The self-intersections inside the polygon are removed by finding the *hull* of the polygon. The hull is itself a polygon that wraps tightly around the other one, compare Figure 4.19 for an illustration. Computing the hull of a polygon is based on the *Graham scan method*, which finds the convex hull of a finite set of points, see [43]. Then, if the hull consists of multiple connected components, the largest of these connected components is chosen as the new polygon and all the smaller components are removed. Afterwards, vertices are added or removed to maintain the total number of vertices.

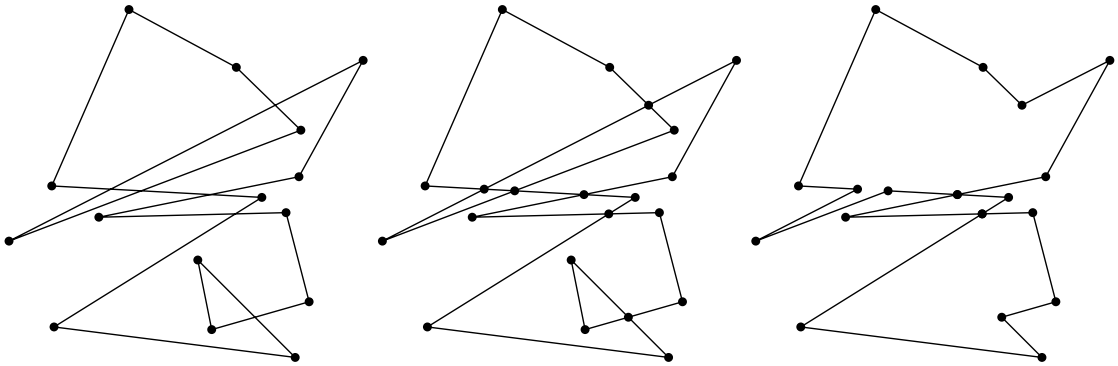

Figure 4.19: A polygon with self-intersections (left), the polygon with the intersections added to its list of vertices (mid) and the polygon hull (right).

In detail, the algorithm consists of the following steps:

First, we need to find all points where edges of the polygon intersect. We can do this by iterating over all edges and looking for intersections with other edges (see Figure 4.19, mid). The intersections are then added to the list of polygon vertices.

Then, by starting with the vertex that has the smallest y-coordinate (it is for sure a vertex of the hull), those line segments are considered that directly connect the

vertex to other vertices or intersections. From these line segments, the one that encloses the smallest angle with the x-axis and its endpoint is chosen as an edge of the hull. This procedure is repeated from the endpoint of that edge, compare Figure 4.20.

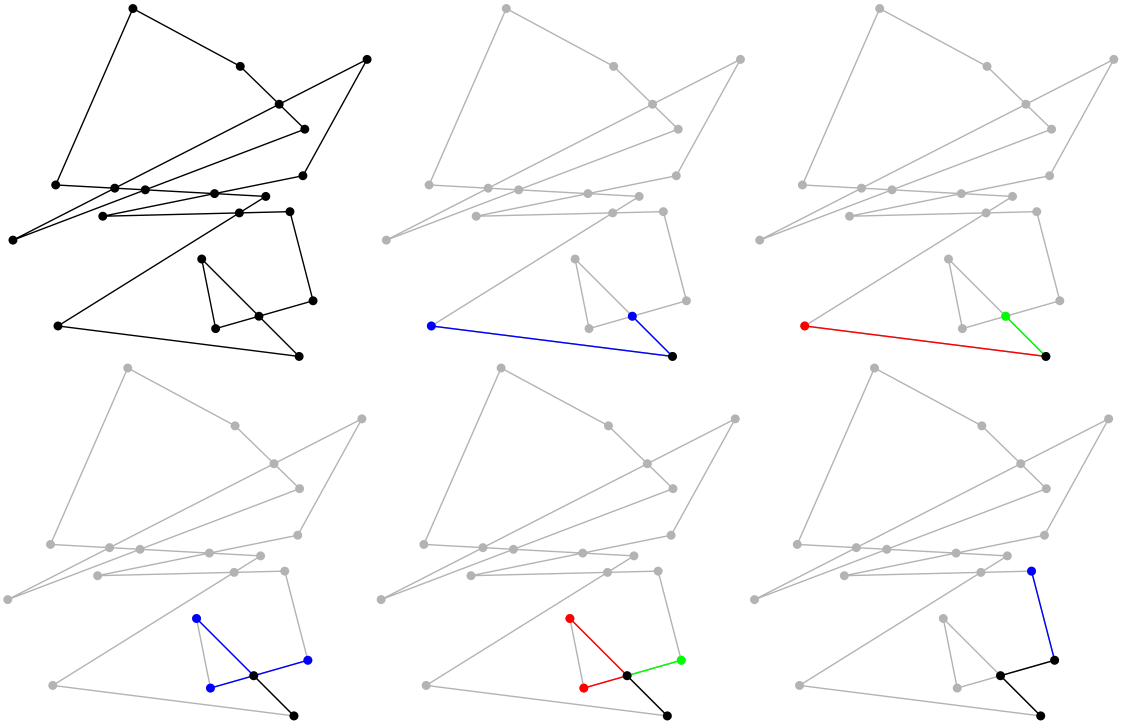

Figure 4.20: Finding the polygon hull. The possible line segments to choose from in each step are blue, the discarded line segments are red, and the chosen segment is green.

When the algorithm arrives at the starting vertex again, it has found the hull of the polygon and terminates, see Figure 4.19, on the right.

Nonetheless, the hull might consist of multiple different connected components. Thus, the polygon hull algorithm is implemented such that the list of vertices of the hull is given as an output. All vertices that appear more than once in the list are points where at least two different connected components are touching. If the polygon consists of more than two connected components, this information can be used to recursively find all connected components of the polygon hull. Then, the largest of the connected components is chosen as the new polygon. Finally, vertices are added as in the relocate vertices step (compare Subsection 4.2.6) to regain the prescribed amount of vertices. We refer to Figure 4.21 for an illustration.

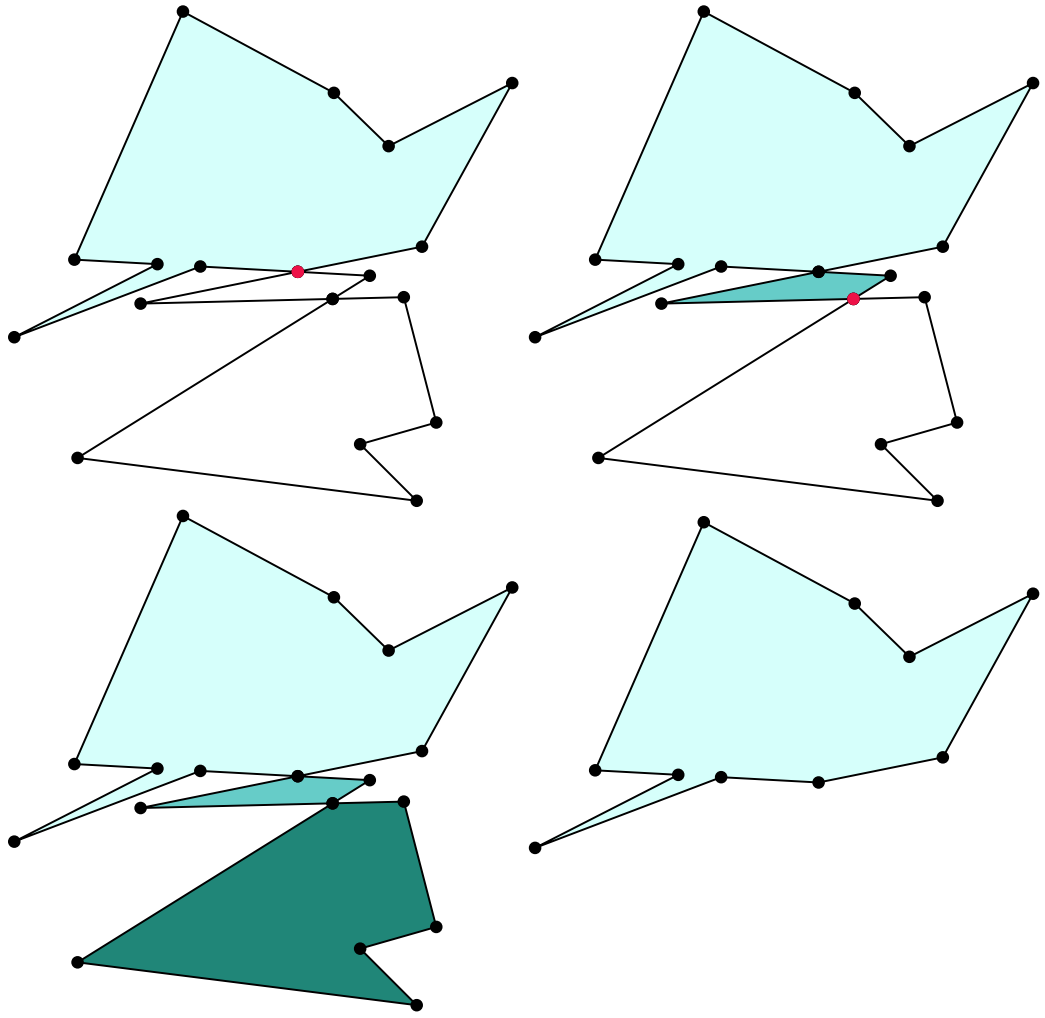

Figure 4.21: Finding and choosing the largest connected component of the polygon's hull.

4.3 Polytope Optimization Algorithm

The steps in the polytope optimization algorithm are very similar to those of the box optimization algorithm and the rotated box optimization algorithm. A flowchart for the most important steps of the polytope optimization algorithm can be found in Figure 4.22.

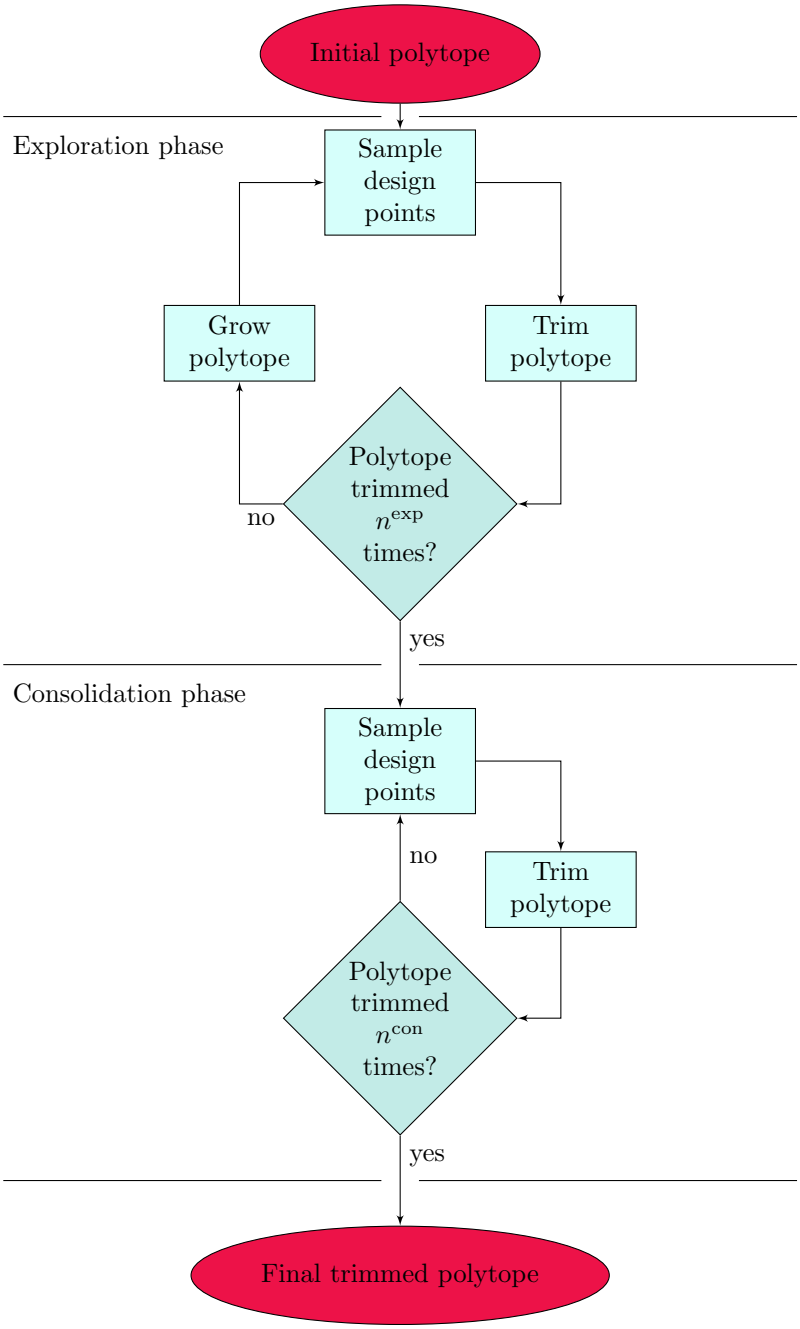

Figure 4.22: A flowchart for the polytope optimization algorithm.

4.3.1 Polytope Initialization

As in the box optimization algorithm and the rotated box optimization algorithm, the initial polytope is usually given or can be constructed around a point found by another algorithm. During the execution of the polytope optimization algorithm, each polygon $P_{i,j}$ of a polytope retains a fixed number M of vertices. Recall from Section 4.2 that a particular polygon $P_{i,j}$ is represented by the ordered sequence of its vertices,

$$P_{i,j} := \left\{ \boldsymbol{v}_{i,j}^{(1)}, \ldots, \boldsymbol{v}_{i,j}^{(M)} \right\}.$$

4.3.2 Exploration Phase

In the *exploration phase*, similar to the box optimization algorithm and the rotated box optimization algorithm, parts of the initial polytope containing bad design points are trimmed. Then, the polytope is grown again. These steps are repeated n^{exp} times. This allows the polytope to move through the design space Ω_{ds} in order to find a spot with a large volume of good design space. After going through all n^{exp} steps of the exploration phase, the algorithm switches to the *consolidation phase*.

Sample Design Points

For each design point $\boldsymbol{x}$, all entries $x_k, k \in \mathcal{J}_{\text{int}}$, and $x_i, x_j, (i,j) \in \mathcal{J}_{\text{pair}}$, are sampled separately. The entries x_k can easily be drawn from the interval I_k. The entries (x_i, x_j) are obtained by sampling a point inside the polygon $P_{i,j}$ as described in Subsection 4.2.1. After the design points are found, they are evaluated and collected in the sets of good design points

$$\mathcal{X}^{\text{good}} := \left\{ {}_{(1)}\boldsymbol{x}^{\text{good}}, \ldots, {}_{(n^{\text{good}})}\boldsymbol{x}^{\text{good}} \right\},$$

and bad design points

$$\mathcal{X}^{\text{bad}} := \left\{ {}_{(1)}\boldsymbol{x}^{\text{bad}}, \ldots, {}_{(n^{\text{bad}})}\boldsymbol{x}^{\text{bad}} \right\},$$

compare Figure 4.23.

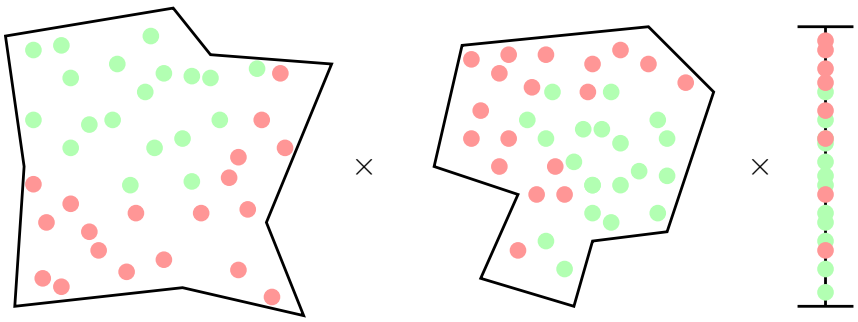

Figure 4.23: A polytope consisting of two polygons and an interval with good and bad design points.

Trim Polytope

After the sampling step, the bad points are removed by trimming the polytope. The framework of this step is outlined in Algorithm 7. It is similar to the trimming algorithm in Subsection 2.1.2. However, due to the complexity added by considering polytopes, it is more involved. As input, a polytope Ω_{pol} and ordered sets of good design points $\mathcal{X}^{\text{good}}$ and bad design points $\mathcal{X}^{\text{bad}}$ (line 1) are required. The output (line 2) is a polytope Ω_{pol} that contains no bad design points.

Algorithm 7 (Trim Polytope). Trim the polytope, keeping as many good design points as possible.

1: **Input:** Ω_{pol}, $\mathcal{X}^{\text{good}}, \mathcal{X}^{\text{bad}}$
2: **Output:** Ω_{pol}

3: **for** $\ell = 1, \ldots, n^{\text{bad}}$ **do**
4: **for** $m = 1, \ldots, n^{\text{good}}$ **do**
5: **for all** $k \in \mathcal{J}_{\text{int}}$ **do**
6: $\Omega^{(k)} \leftarrow \texttt{trim\_interval}\Big(\Omega_{\text{pol}}, k, {}_{(m)}\boldsymbol{x}^{\text{good}}, {}_{(\ell)}\boldsymbol{x}^{\text{bad}}\Big)$
7: **end for**
8: **for all** $(i,j) \in \mathcal{J}_{\text{pair}}$ **do**
9: $\Omega_{i,j} \leftarrow \texttt{trim\_polygon}\Big(\Omega_{\text{pol}}, (i,j), {}_{(m)}\boldsymbol{x}^{\text{good}}, {}_{(\ell)}\boldsymbol{x}^{\text{bad}}\Big)$
10: **end for**
11: $\Omega^{(\ell)} \leftarrow \texttt{evaluate}\Big(\big\{\Omega^{(k)}\big\}_{k\in\mathcal{J}_{\text{int}}}, \big\{\Omega_{i,j}\big\}_{(i,j)\in\mathcal{J}_{\text{pair}}}\Big)$
12: **end for**
13: $\Omega_{\text{pol}} \leftarrow \texttt{evaluate}\Big(\big\{\Omega^{(\ell)}\big\}_{\ell=1}^{n^{\text{good}}}\Big)$
14: **end for**
15: $\Omega_{\text{pol}} \leftarrow \texttt{reshape}\big(\Omega_{\text{pol}}\big)$

Because the bad design points have to be removed successively, a loop over the bad design points is initialized in line 3. Since as many good design points as possible should be kept, the good design points are iterated and for each iterated good design point ${}_{(m)}\boldsymbol{x}^{\text{good}}$, the polytope is trimmed such that at least ${}_{(m)}\boldsymbol{x}^{\text{good}}$ is kept inside it (line 4). Note that this order of iterations is different from the trimming algorithm in Subsection 2.1.2, where we iterated over the good designs first. We changed the order to increase the stability of the algorithm, as shown in Section 5.1.

For each iterated good design point ${}_{(m)}\boldsymbol{x}^{\text{good}}$, the current bad design point ${}_{(\ell)}\boldsymbol{x}^{\text{bad}}$ is removed from the polytope such that at least ${}_{(m)}\boldsymbol{x}^{\text{good}}$ remains within the polytope. The bad design point ${}_{(\ell)}\boldsymbol{x}^{\text{bad}}$ is removed by moving the boundary of an interval I_k or a polygon $P_{i,j}$ onto ${}_{(\ell)}\boldsymbol{x}^{\text{bad}}$, thereby trimming the polytope. Thus, all intervals I_k and polygons $P_{i,j}$ are iterated (see lines 5–10). For each interval I_k and polygon $P_{i,j}$, the boundary is moved onto ${}_{(\ell)}\boldsymbol{x}^{\text{bad}}$ via the `trim_interval` and `trim_polygon` algorithms and the resulting polytope is stored in a new variable $\Omega^{(k)}$ or $\Omega_{i,j}$, respectively, leaving all other intervals and polygons untouched. Note that the algorithm `trim_polygon` coincides with Algorithm 5 except for needing the coordinates (i,j) of the respective polygon $P_{i,j}$ as input arguments. The algorithm `trim_interval` is the same as lines 7–11 from Algorithm 1. It operates on the interval I_k and simply

relocates one of the end points onto the bad design point such that the good design point remains in the output interval.

Then, in line 11, the function `evaluate` is applied to all polytopes $\Omega^{(k)}$ and $\Omega_{i,j}$. It returns the result $\Omega^{(\ell)}$ that maximizes the quality measures, applied in the same order as listed in Subsection 4.2.4. The quality measures are modified for polytopes such that the polytope with the most polygons that fulfill the self-intersection and angle-size measures that also contains as many good design points as possible is chosen as the optimum.

After every good point has been iterated once, the polytopes $\Omega^{(\ell)}$ (line 13) are evaluated and the best of them is used to replace Ω_{pol}. Following this, the next iteration of the loop starts, where the next bad design point is removed. The polygon trimming is completed when all of the bad points are removed.

In order to avoid degenerate polytopes, spikes are finally removed and vertices are relocated by the function `reshape` in line 15. This function consists of the operations "Remove Spikes" (see Subsection 4.2.5) and "Relocate Vertices" (see Subsection 4.2.6), which are applied to each polygon $P_{i,j}$ individually as explained in Subsections 4.2.5 and 4.2.6, respectively.

Grow Polytope

The polytope is grown as the final operation of a single step of the exploration phase. Therefore, the end points a_k and b_k of each interval I_k are moved by a factor $g^{(\ell)}$ in order to grow the polytope in each dimension k. Each polygon $P_{i,j}$ is grown by the factor $g^{(\ell)}$ as explained in Subsection 4.2.7. Here, the factor $g^{(\ell)}$ is the growth rate and can be calculated as described in Subsection 2.1.2. If necessary, a polygon $P_{i,j}$ that lies partially outside of the design space Ω_{ds} is retracted as described in Subsection 4.2.8.

4.3.3 Consolidation Phase

Having completed the *exploration phase*, the candidate polytope is handed over to the consolidation phase. Similar to the box optimization algorithm and the rotated box optimization algorithm, one step of the consolidation phase consists of the steps "Sample Design Points" and "Trim Polytope" (see Subsection 4.3.2) but the polytope is no longer grown. We still remove spikes that might be introduced by the trimming. However, we no longer relocate vertices because we cannot control whether we relocate vertices that lie in good design space. This would mean that we could unnecessarily remove good design space that we cannot regain because the polytope is no longer grown. The consolidation phase is terminated after a fixed number of n^{con} steps. It is terminated earlier when no bad design points have been sampled three times in series. The resulting polytope is returned as the final output of the polytope algorithm.

4.4 Numerical Experiments

In this section, we apply the polytope optimization algorithm to the same toy problems that we used for the box optimization algorithm and the rotated box optimization algorithm. Afterwards, we conduct a parameter study that explores various settings for the polytope optimization algorithm.

4.4.1 Two Example Problems in 2D

We consider the polygon and Rosenbrock problems (see Subsection 2.4.1) again, now for the polytope optimization algorithm. The settings are similar to those applied for the box optimization algorithm and the rotated box optimization algorithm. The polytope optimization algorithm is carried out 100 times for each problem. Every time, the number of steps in the exploration and the consolidation phase is set to $n^{\text{exp}} = n^{\text{con}} = 100$. The trimming strategy in the exploration phase is "Edge Walking". Additionally, in every step, 100 design points are sampled. The growth rate is dynamic with $g^{(0)} = 0.05$ and $a^{\text{target}} = 0.6$. The polytope optimization algorithm is performed with polygons that have $M = 10$ vertices, where the required minimum size of the angles is $\alpha = 20°$ and the vertex relocation takes place in every tenth step of the exploration phase.

2D Polygon

As for the rotated box optimization algorithm, the 2D-map is $\Omega_{1,2} := [0, 4] \times [0, 4]$. The vertices of the initial polytope are given by

$$\boldsymbol{v}^{(i)} := \begin{pmatrix} 1.3 + 0.2 \cdot \cos(i \cdot 2\pi/10 + \pi/4) \\ 1.8 + 0.2 \cdot \sin(i \cdot 2\pi/10 + \pi/4) \end{pmatrix}$$

with $i = 1, \ldots, M$, i.e., they lie on a circle with center $(1.3, 1.8)^\top$ and radius 0.2 (see also Figure 4.24).

The mean normalized volume of the 100 polytopes is 0.256 with a standard deviation of 0.019. This amounts to 76% more volume than found by the box optimization algorithm and 36% more volume than found by the rotated box optimization algorithm. Thus, the mean normalized volume of the polytopes comes close to the volume of the polygonal good design space, which is 0.2959. Additionally, the polytopes can take on the shape of the good design space, as can be seen in Figure 4.24.

Rosenbrock Function

For the Rosenbrock function, the 2D-map is again $\Omega_{1,2} := [-2, 2] \times [-2, 3]$. The initial polytope is given by

$$\boldsymbol{v}^{(i)} := \begin{pmatrix} 1.3 + 0.2 \cdot \cos(i \cdot 2\pi/10 + \pi/4) \\ 1.8 + 0.2 \cdot \sin(i \cdot 2\pi/10 + \pi/4) \end{pmatrix}$$

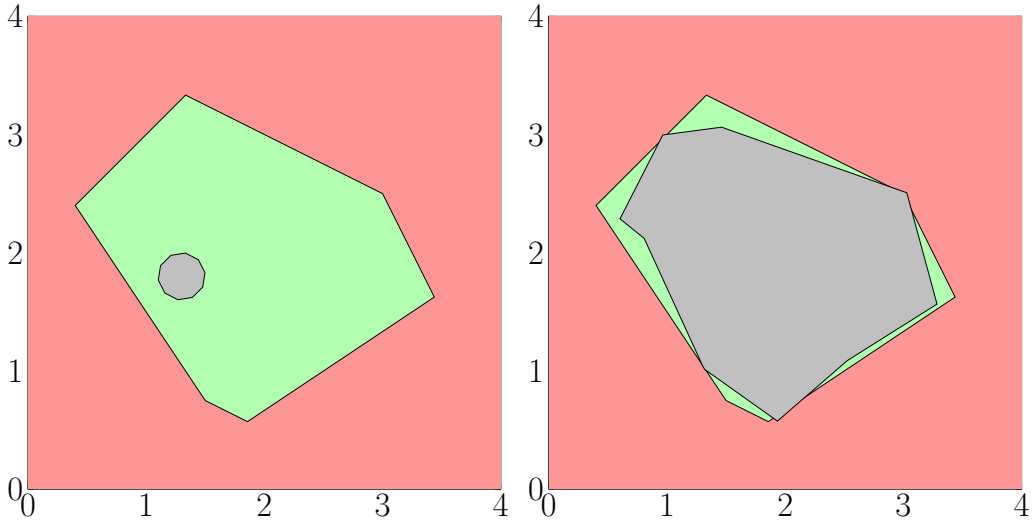

Figure 4.24: The initial polytope (left) and a polytope with normalized volume 0.2567 (right) for the 2D polygon problem.

with $i = 1, \ldots, M$, see Figure 4.25. The mean normalized volume after 100 executions is 0.0828 with a standard deviation of 0.0178. This amounts to 259% more volume than found by the box optimization algorithm and 249% more volume than found by the rotated box optimization algorithm. As can be see in Figure 4.25, the polytope finds the lower half of the U-shaped good design space. Notably, it settles on both sides of the U-shape which means that the algorithm is able to move around corners while keeping a non-convex shape.

In conclusion, the results suggest that polytope optimization algorithm works well. It is also able to adapt to the good design space better than the box optimization algorithm and the rotated box optimization algorithm, as the shape of a polytope is rather flexible.

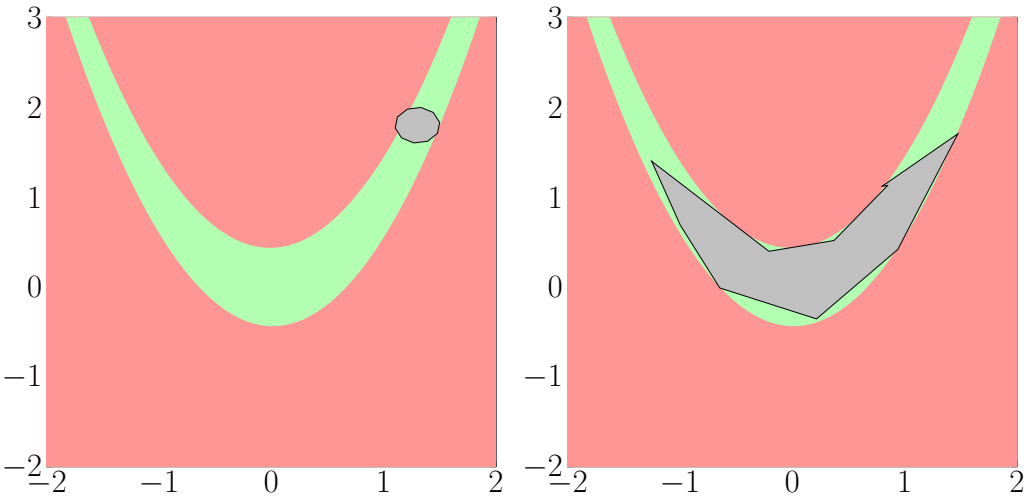

Figure 4.25: The initial polytope (left) and a polytope with normalized volume 0.0837 (right) for the Rosenbrock problem.

4.4.2 Parameter Studies

The polytope optimization algorithm involves new parameters that affect it in different ways: the number of vertices for the initial polytope, the minimum/maximum angle inside the polytope, the trimming strategy and the number of times one or more vertices should be relocated. As it is unclear what the impact of the parameters on the algorithm is, we conduct a parameter study that analyzes all of them. For each study, we use the same parameter settings as in the previous subsection:

- We consider the Rosenbrock function for the whole parameter study.
- There are 100 executions of the algorithm.
- The initial polytope is given by

$$\boldsymbol{v}^{(i)} := \begin{pmatrix} 1.3 + 0.2 \cdot \cos(i \cdot 2\pi/10 + \pi/4) \\ 1.8 + 0.2 \cdot \sin(i \cdot 2\pi/10 + \pi/4) \end{pmatrix},$$

  for $i = 1, \ldots, M$.
- The exploration and consolidation phases consist of $n^{\text{exp}} = n^{\text{con}} = 100$ steps.
- In every step, $N = 100$ design points are sampled.
- The growth rate is dynamic with $a^{\text{target}} = 0.6$ and $g^{(0)} = 0.05$.
- Each polygon on a 2D-map has $M = 10$ vertices.
- The trimming strategy is "Edge Walking".
- The required minimum size of angles is $\alpha = 20°$.
- The vertex relocation takes place in every tenth step of the exploration phase, relocating the shortest edge.

However, for each study, we vary one of the four new parameters and observe the outcome.

Number of Vertices

In a first study, we vary the number of vertices. We start at $M = 4$ vertices, as the results should then be similar to the rotated box optimization algorithm, and go through $M = 10$, 15, 25 and 40. The results can be found in Table 4.1.

Clearly, the mean normalized volume for polytopes with 4 vertices is similar to that of the box optimization algorithm, see Subsection 3.4.1. Also, for more than 15 vertices, the volume does not increase. Thus, more vertices do not lead to a larger volume.

We have created heat maps to analyze how well the U-shaped good design space is approximated by all polytopes in Figure 4.26. In a heat map, a region that is covered by almost every polytope is displayed in white-yellow, while a region that is covered by very few polytopes is displayed in dark red. A region that is black is not

M	Mean	Standard Deviation
4	0.0361	0.0069
10	0.0828	0.0178
15	**0.096**	**0.0125**
25	0.0914	0.0142
40	0.0925	0.0114

Table 4.1: Mean and standard deviation for the normalized volume of 100 polytopes calculated with varying numbers of vertices. The best result is printed in bold.

covered by any polytope. For polytopes with 4 vertices, the polytope optimization algorithm behaves similar to the box optimization algorithm and settles somewhere at the bottom of the U-shape. For 10 vertices, the algorithm struggles to find the left region of the U-shape, probably because the initial polytope is in the right-hand region. For a higher number of vertices, the polytope optimization algorithm is able to cover most of the U-shape, although it is still biased slightly towards the right region. A number of 15 vertices seems to be enough to gain good results from the polytope optimization algorithm.

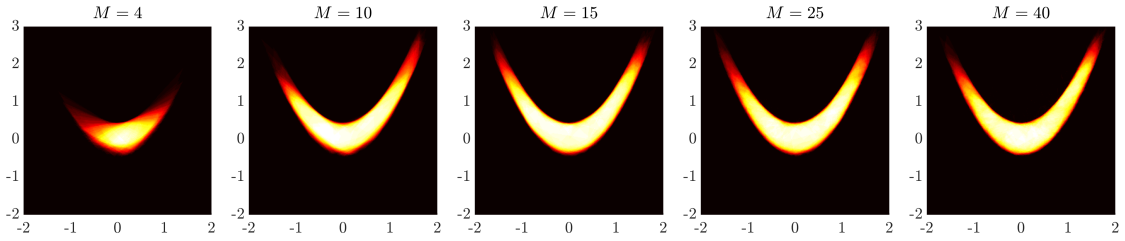

Figure 4.26: From left to right: Heat maps for $M = 4$, 10, 15, 25 and 40 vertices.

In Figure 4.27, for each number of vertices, we have given a polytope with a volume near the mean normalized volume of polytopes with that number of vertices. Note that the polytope for $M = 4$ has indeed four vertices, one of its angles is simply close to 180°. For $M = 10$ and $M = 15$, the polytopes approximate the U-shape reasonably well. For $M = 25$ and $M = 40$, one can see sharp spikes jutting from the polytopes. This is an effect that occurs when there are many vertices. However, upon closer inspection (see Figure 4.28), these spikes turn out not to be spikes in the sense of the "Minimum/Maximum Size of Angles" criterion because all the angles have a valid size. Rather, they are very thin branches. These appear in the exploration phase when very short edges are trimmed unfavorably and then pushed further outside in a growth step. If we allow vertex relocations in the consolidation phase, the branches might disappear: at least one of the edges of a branch should be very short and if that edge is relocated, it is transformed to a vertex. That vertex would then have a very small angle and thus it might be seen as a spike which consequently would be removed. We investigate this hypothesis later in this subsection.

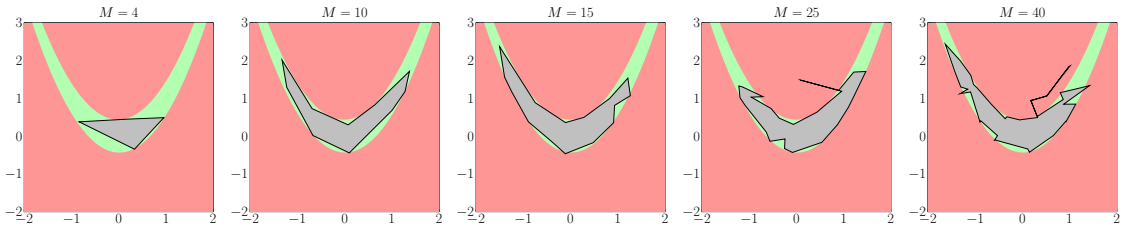

Figure 4.27: From left to right and top to bottom: Polytopes close to the mean for $M = 4$, 10, 15, 25 and 40 vertices.

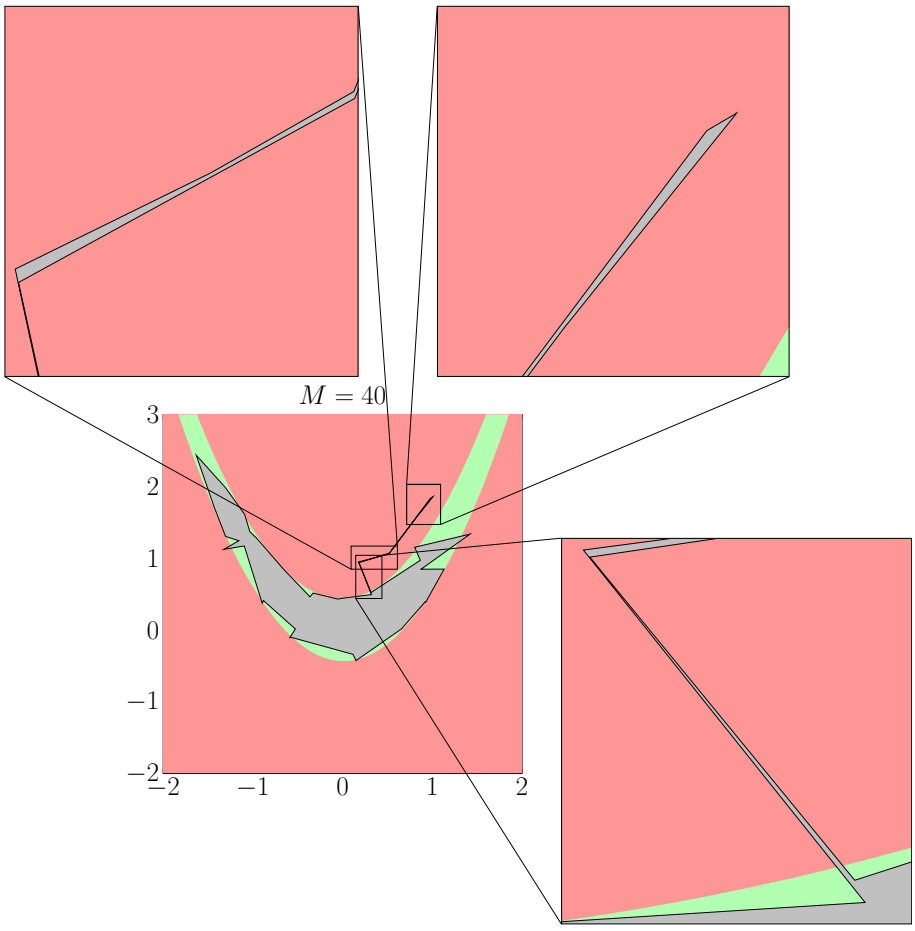

Figure 4.28: A branch formed by a polytope with 40 vertices.

Minimum Angle

We study the size of the minimum/maximum angle, letting $\alpha = 1°$, $5°$, $10°$, $20°$, $30°$, $40°$, $50°$ and $60°$. The resulting mean normalized volumes can be found in Table 4.2.

α	Mean	Standard Deviation
1°	**0.0949**	**0.0080**
5°	0.0944	0.0147
10°	0.0918	0.0157
20°	0.0828	0.0178
30°	0.0822	0.0115
40°	0.0757	0.0129
50°	0.0726	0.0161
60°	0.0696	0.0141

Table 4.2: Mean and standard deviation for the normalized volume of 100 polytopes calculated with varying minimum angles.

For an increasing angle α, the mean normalized volume decreases and the polytopes get stuck in the bottom of the U-shape (see Figure 4.29). A look at Figure 4.30 explains why: polytopes with small angles are able to stretch far into the left and right regions of the U-shape. Only two or three vertices with sharp angles are necessary to move into the good design space on the left and on the right, the remaining vertices are used to fill out the curve at the bottom. Polytopes with larger angles cannot form the same spikes and thus cannot move into the sides as easily.

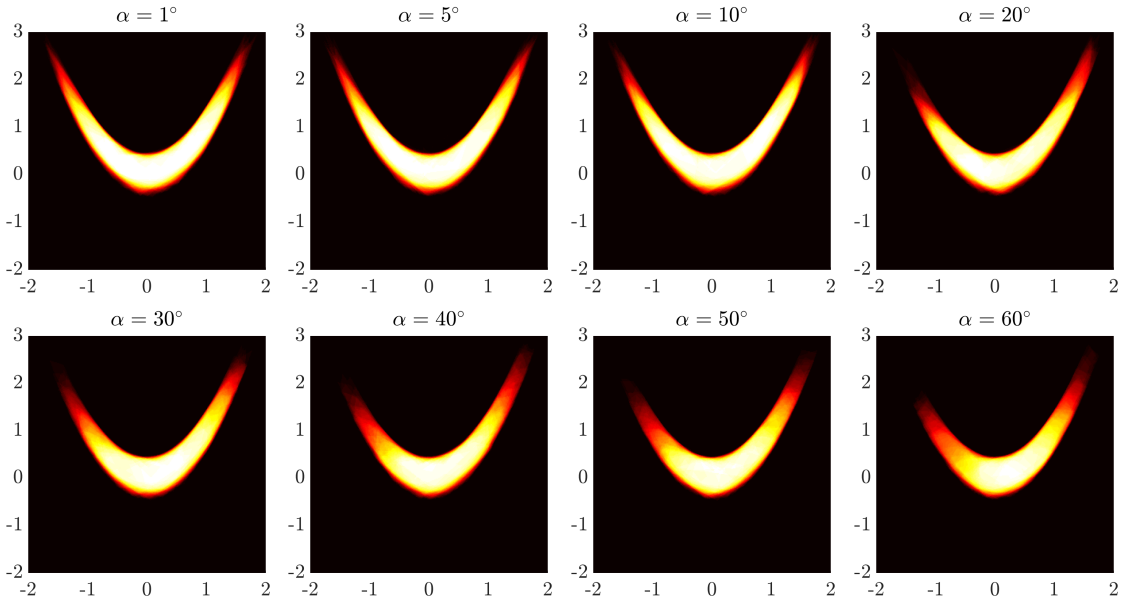

Figure 4.29: From left to right and top to bottom: Heat maps for $\alpha = 1°$, $5°$, $10°$, $20°$, $30°$, $40°$, $50°$ and $60°$.

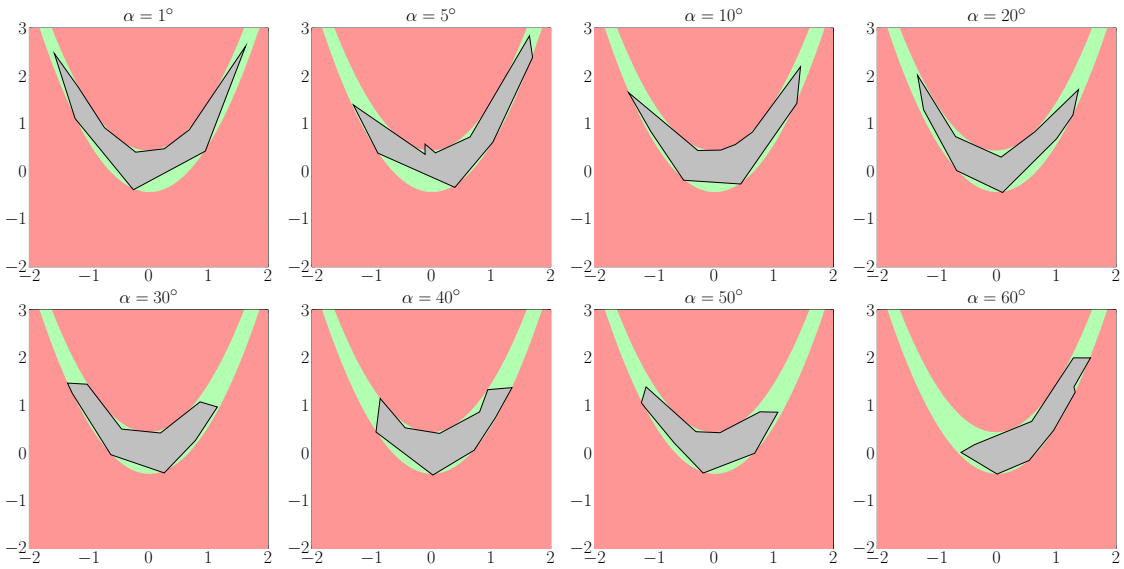

Figure 4.30: From left to right and top to bottom: Polytopes close to the mean for $\alpha = 1°$, $5°$, $10°$, $20°$, $30°$, $40°$, $50°$ and $60°$.

Trimming Strategies

Next, we examine the three trimming strategies "Edge Walking", "Move Edge" and "Move Vertex". The mean normalized volumes can be found in Table 4.3. These values and the heat maps in Figure 4.31 show that the "Edge Walking" strategy is superior to the other two strategies. Notably, the algorithm failed six times during the execution of the "Move Vertex" strategy, meaning that at one point the polytope was trimmed such that it contained only bad designs after the next sampling step.

The success of "Edge Walking" over the other two strategies might be explained as follows: The "Move Vertex" strategy moves vertices directly onto bad points, thus the polytope will, to some extent, try to approximate the border of the bad design space. This effect can be seen in Figure 4.32, where all vertices lie close to the bad design space and the polytope especially clings to the upper border of the bad design space. This effect is visible for the "Move Edge" strategy, too, although not as pronounced. In contrast, for the "Edge Walking" strategy, none of the vertices touch the upper boundary of the bad design space. Rather, only the edges of the polytope touch it, staying tangent to the upper half. This means that the bad design space is not approximated as exactly as for the other two strategies, and the algorithm applies less vertices to this task. In return, the algorithm can use these free vertices to explore a larger part of the good design space than with the other strategies, which results in a larger polytope.

Now, why is the "Edge Walking" strategy able to do this while the "Move Edge" strategy is not? After all, both strategies operate similarly. To answer this question, we refer back to Figures 4.9 and 4.11. On these pictures, we can see that the vertices get pushed further into potential good design space with the "Edge Walking" strategy than with the "Move Edge" strategy, where the vertices stay closer to the border of the potential bad design space. Figure 4.33 clarifies this effect.

Through repeated trimming, all vertices are pushed into the good design space such

that their associated edges only form tangents or secants with the border of the bad design space. By trimming repeatedly with the "Move Edge" strategy, the vertices, and thus the edges, will stay close to the border, thereby trying to approximate it and losing a lot of flexibility.

Strategy	Mean	Standard Deviation
Edge Walking	**0.0828**	**0.0178**
Move Edge	0.0601	0.0080
Move Vertex	0.0412	0.0113

Table 4.3: Mean and standard deviation for the normalized volume of 100 polytopes calculated with the three different trimming strategies.

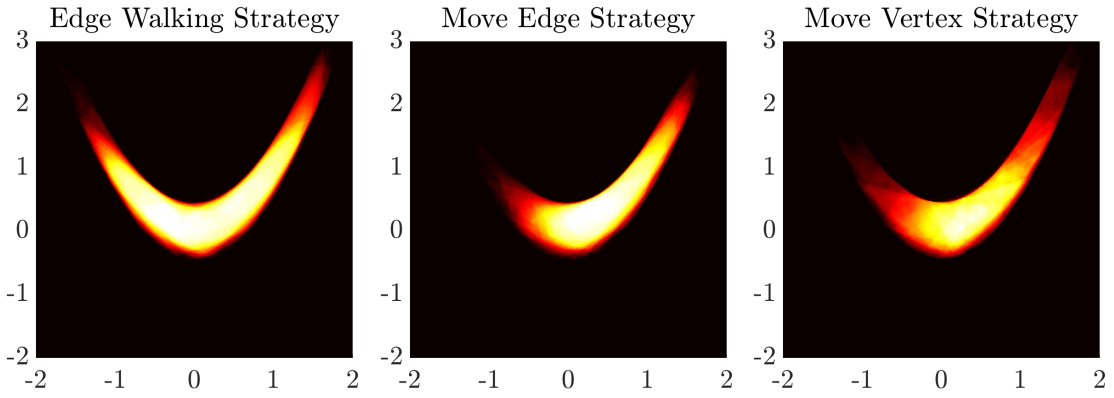

Figure 4.31: From left to right: Heat maps for the "Edge Walking", "Move Edge" and "Move Vertex" strategies.

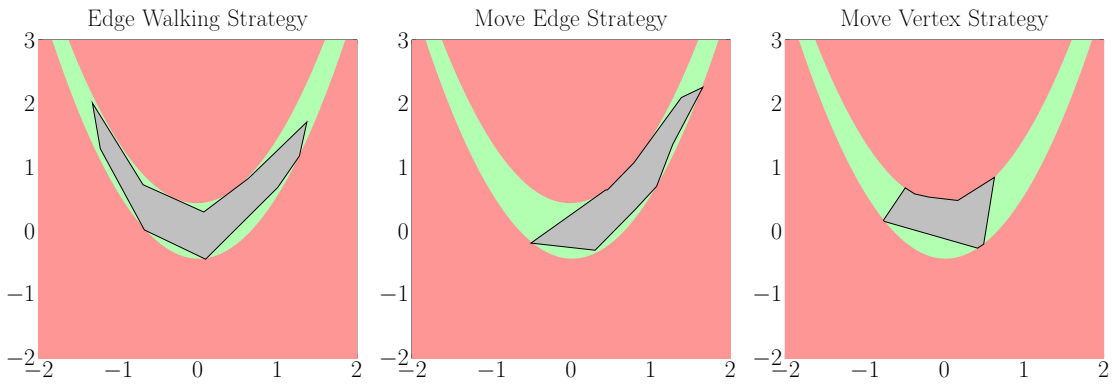

Figure 4.32: From left to right: Polytopes close to mean for the "Edge Walking", "Move Edge" and "Move Vertex" strategies.

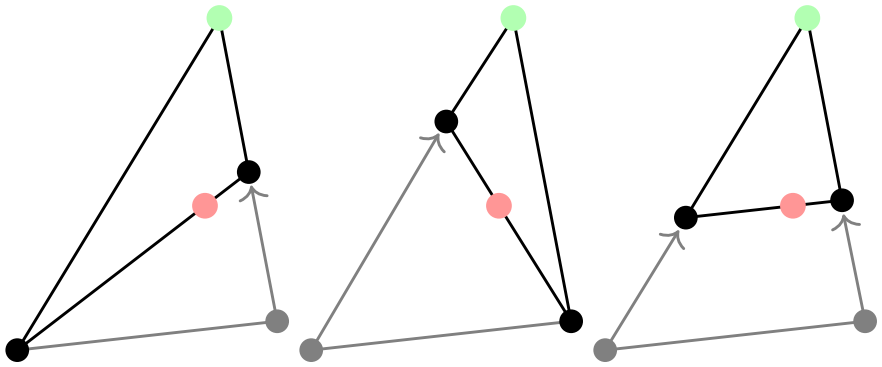

Figure 4.33: Vertices get pushed further into potential good design space with the "Edge Walking" strategy (left and middle) than with the "Move Edge" strategy (right).

Another question arises: How can the algorithm fail when we apply the "Move Vertex" strategy? The algorithm can only fail if at some point only bad design points are sampled. Then, the algorithm can no longer trim the polyotpe in a meaningful way and quits. A polytope for which this happens is shown in Figure 4.34, top left. The polytope is already in the consolidation phase, i.e., it gets only trimmed further and does no longer grow. It is trimmed such that it forms a thin branch (Figure 4.34, top right). While the trimming may seem counterintuitive, this is the only valid outcome for this polytope. For all other trimming possibilities, the angle criterion would have failed. The branch does not get removed in subsequent steps of the exploration phase (Figure 4.34, bottom) and, after a few trimming steps, reaches the state shown in the left picture of Figure 4.35.

Note that multiple vertices are scattered on the thin branch to the left. After the last trimming step, spikes are removed from the polytope. With this action, the spike to the left is shortened, but not removed, as the spike consists of multiple vertices. However, the large spike on the bottom has an angle just small enough that it gets removed by the algorithm. The resulting polytope lies completely in the bad design space and, in the next sampling step, only bad design points will be sampled, causing the algorithm to fail.

We might be able to produce better results if we somehow were able to remove branches from a polytope. One possible way to do this is by allowing vertex relocation in the consolidation phase. Because the end of a branch consists of a very small edge, vertex relocation might replace that edge with a single vertex. Thus, the end of the branch would be turned into a spike, which would get removed automatically by a "Remove Spikes" step. We investigate this procedure in the following parameter studies.

In conclusion, the "Edge Walking" strategy seems to be the most flexible strategy. It yields the polytopes with the largest volume and discovers more good design space than the other two strategies. While they might be viable when vertices are relocated (see the following parameter studies), the "Edge Walking" strategy yields good results without having to modify it and is thus recommended as the standard trimming strategy for the polytope optimization algorithm.

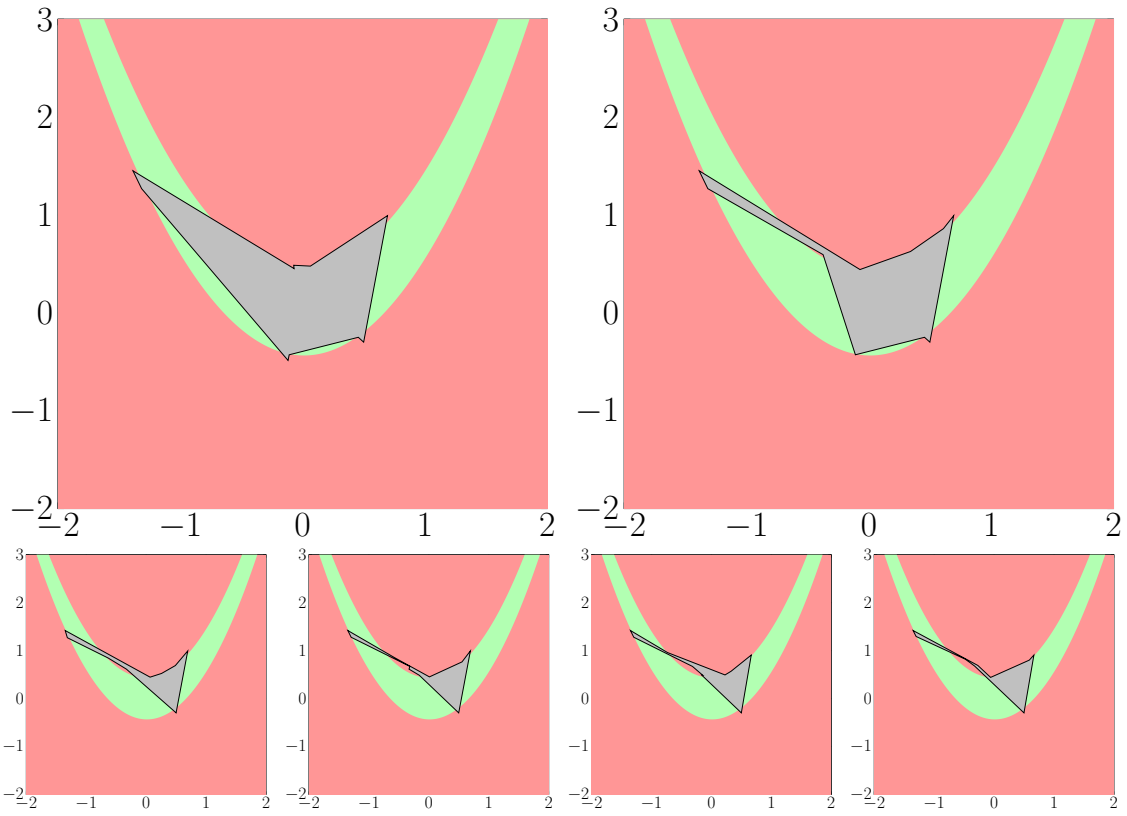

Figure 4.34: A branch being introduced to a polytope with the "Move Vertex" strategy (top pictures). It does not get removed in subsequent steps (bottom pictures).

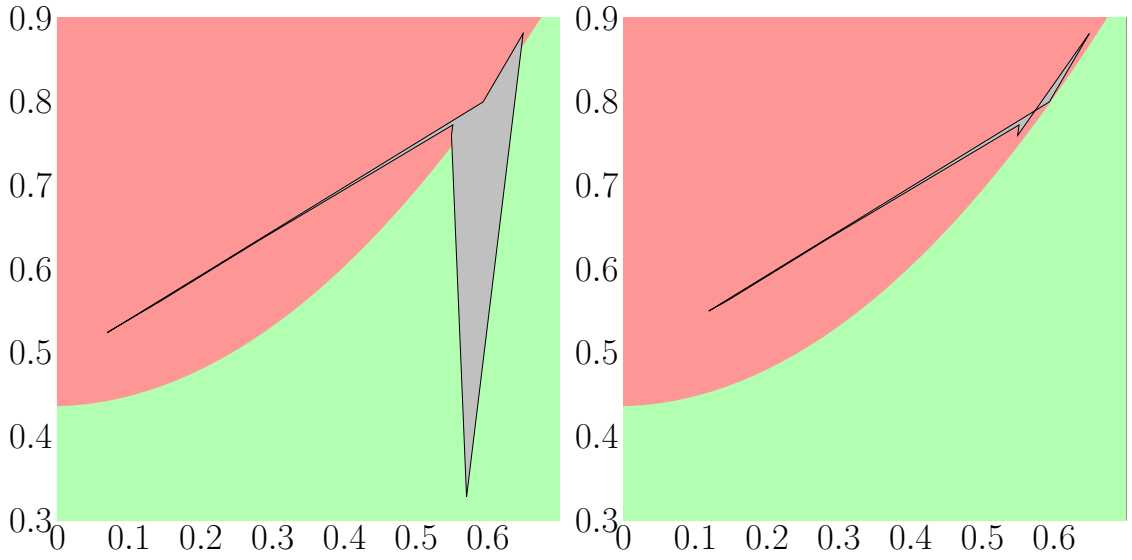

Figure 4.35: Spikes are removed from a polytope and the result lies completely in the bad design space. The result is split into two connected components.

Vertex Relocation

We consider multiple configurations for the "Relocate Vertices" step. Every first, third, fifth and tenth step of the exploration phase, we relocate 0, 1, 2 or 4 vertices. The results for each of these combinations can be found in Table 4.4. Apparently, relocating only one or two vertices does not make much of a difference in the volume. However, relocating four vertices is detrimental to the volume.

Additionally, we have plotted 100 polytopes for each of the configurations of the "Relocate Vertices" step in Figures 4.36 and 4.37. We prefer this visualization over the visualization with heat maps because it is easier to see how many degenerate spikes or branches outside of the good design space are formed by the polytopes. There are almost no spikes or branches present in these pictures, although there are already a few spikes and branches when we do not relocate vertices. In conclusion, the "Relocate Vertex" step seems to have only a minor impact on the shape of the polytopes. The only exception is when we relocate four vertices in every step (see bottom left of Figure 4.37). As the polytope has only ten vertices, relocating almost half of them changes the shape of the polytope too much, reducing its size significantly.

Relocate k Vertices	Relocate Every ℓ-th Step	Mean	Standard Deviation
$k = 0$	–	0.0859	0.0130
$k = 1$	$\boldsymbol{\ell = 1}$	**0.0886**	**0.0113**
	$\ell = 3$	0.0884	0.0105
	$\ell = 5$	0.0865	0.0137
	$\ell = 10$	0.0874	0.0144
$k = 2$	$\ell = 1$	0.0857	0.0090
	$\boldsymbol{\ell = 3}$	**0.0889**	**0.0099**
	$\ell = 5$	0.0852	0.0132
	$\ell = 10$	0.0847	0.0131
$k = 4$	$\ell = 1$	0.0648	0.0099
	$\boldsymbol{\ell = 3}$	**0.0829**	**0.0133**
	$\ell = 5$	0.0766	0.0116
	$\ell = 10$	0.0752	0.0154

Table 4.4: Mean and standard deviation of the normalized volume for 100 polytopes calculated with multiple settings for "Relocate Vertices".

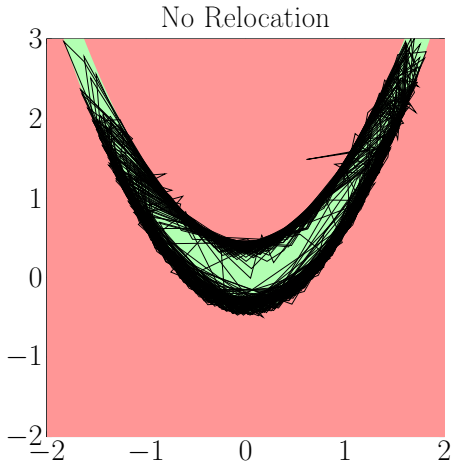

Figure 4.36: 100 polytopes without a “Relocate Vertices” step.

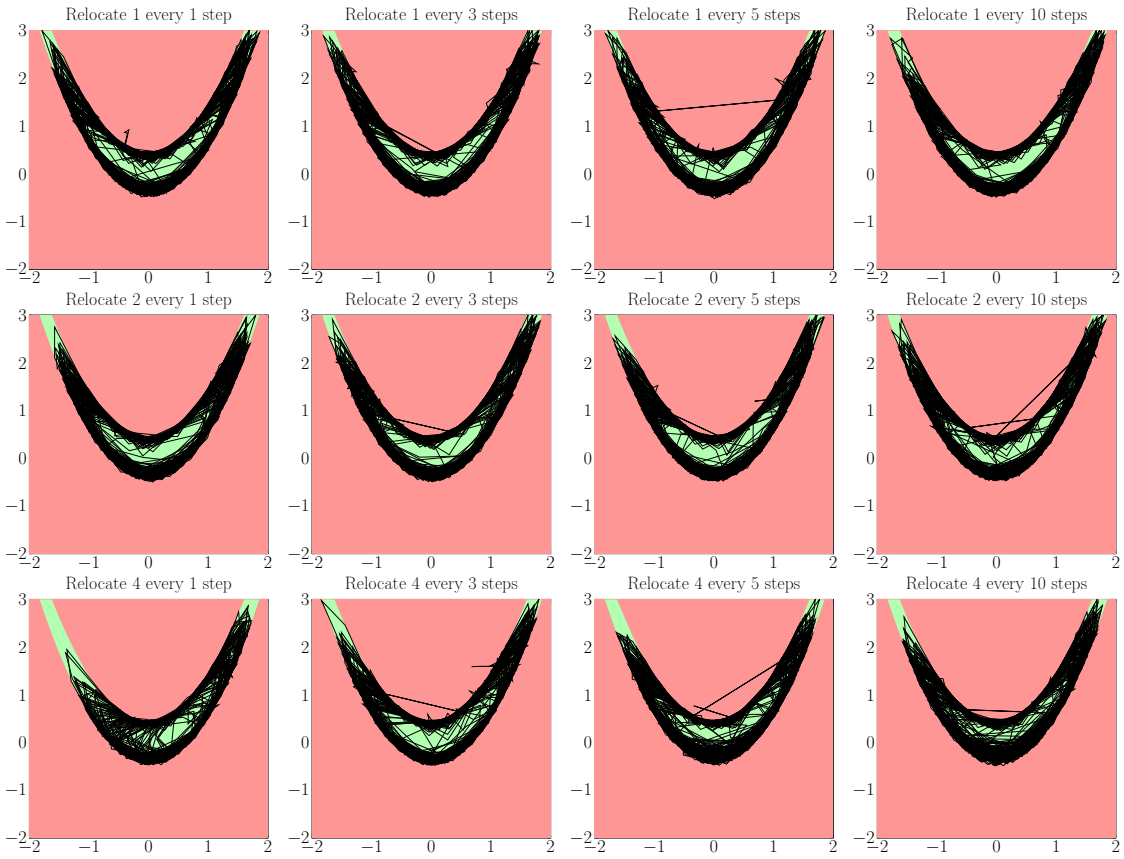

Figure 4.37: 100 polytopes for different configurations of the “Relocate Vertices” step.

Vertex Relocation for "Move Edge" and "Move Vertex"

Previously, we have observed that a stronger vertex relocation might help improve the "Move Edge" and "Move Vertex" strategies if the relocation is also applied in the consolidation phase. Thus, we apply these strategies to the polytope optimization algorithm with the configurations for the "Relocate Vertices" step from before, except that we relocate vertices in the consolidation phase, too.

The results for "Move Vertex" and "Move Edge" can be found in Tables 4.5 and 4.6. We can gather from there that the volume does not increase compared to Table 4.3. Also, when we do not relocate vertices, we gain a higher volume than when we do. We plotted the 100 polytopes calculated by each configuration in Figures 4.38 and 4.39 for the "Move Vertex" strategy and in Figures 4.40 and 4.41 for the "Move Edge" strategy. Again, very few branches and spikes are visible when we do not relocate vertices, and even fewer are visible when we do so. We thus conclude that the vertex relocation has only little impact on the shape of the polytopes. Additionally, when we apply the "Move Vertex" strategy, the algorithm still sometimes fails during its execution. Hence, a stronger vertex relocation does not help to improve the shortcomings of the "Move Vertex" and "Move Edge" strategies.

Relocate k Vertices	Relocate Every ℓ-th Step	Mean	Standard Deviation	Fails
$k = 0$	–	0.0405	0.0151	1
$k = 1$	$\ell = 1$	0.0328	0.0092	2
	$\ell = 3$	0.0330	0.0134	0
	$\boldsymbol{\ell = 5}$	**0.0355**	**0.0142**	**0**
	$\ell = 10$	0.0351	0.0152	3
$k = 2$	$\ell = 1$	0.0235	0.0078	0
	$\ell = 3$	0.0313	0.0102	0
	$\ell = 5$	0.0348	0.0120	1
	$\boldsymbol{\ell = 10}$	**0.0379**	**0.0116**	**3**
$k = 4$	$\ell = 1$	0.0141	0.0059	0
	$\ell = 3$	0.0213	0.0090	2
	$\ell = 5$	0.0238	0.0098	4
	$\boldsymbol{\ell = 10}$	**0.0315**	**0.0112**	**2**

Table 4.5: Mean and standard deviation of the normalized volume for 100 polytopes and number of failures of the polytope optimization algorithm for the strategy "Move Vertex" and multiple settings of "Relocate Vertices".

Relocate k Vertices	Relocate Every ℓ-th Step	Mean	Standard Deviation
$k = 0$	–	0.0605	0.0086
$k = 1$	$\ell = 1$	0.0435	0.0081
	$\ell = 3$	0.0547	0.0072
	$\ell = 5$	0.0558	0.0073
	$\boldsymbol{\ell = 10}$	**0.0579**	**0.0092**
$k = 2$	$\ell = 1$	0.0265	0.0092
	$\ell = 3$	0.0431	0.0096
	$\ell = 5$	0.0488	0.0082
	$\boldsymbol{\ell = 10}$	**0.0537**	**0.0100**
$k = 4$	$\ell = 1$	0.0178	0.0056
	$\ell = 3$	0.0308	0.0095
	$\ell = 5$	0.0391	0.0095
	$\boldsymbol{\ell = 10}$	**0.0482**	**0.0102**

Table 4.6: Mean and standard deviation of the normalized volume for 100 polytopes calculated with the "Move Edge" strategy and multiple settings for "Relocate Vertices".

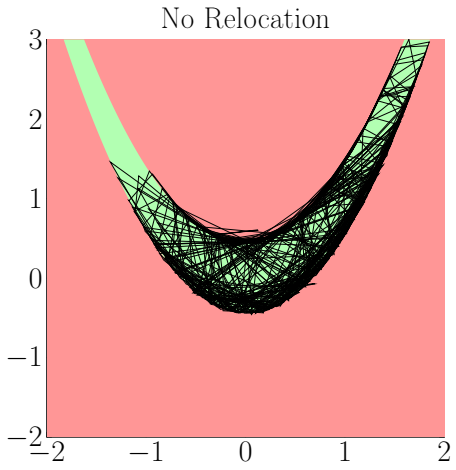

Figure 4.38: 100 polytopes calculated with the "Move Vertex" strategy and no vertex relocation.

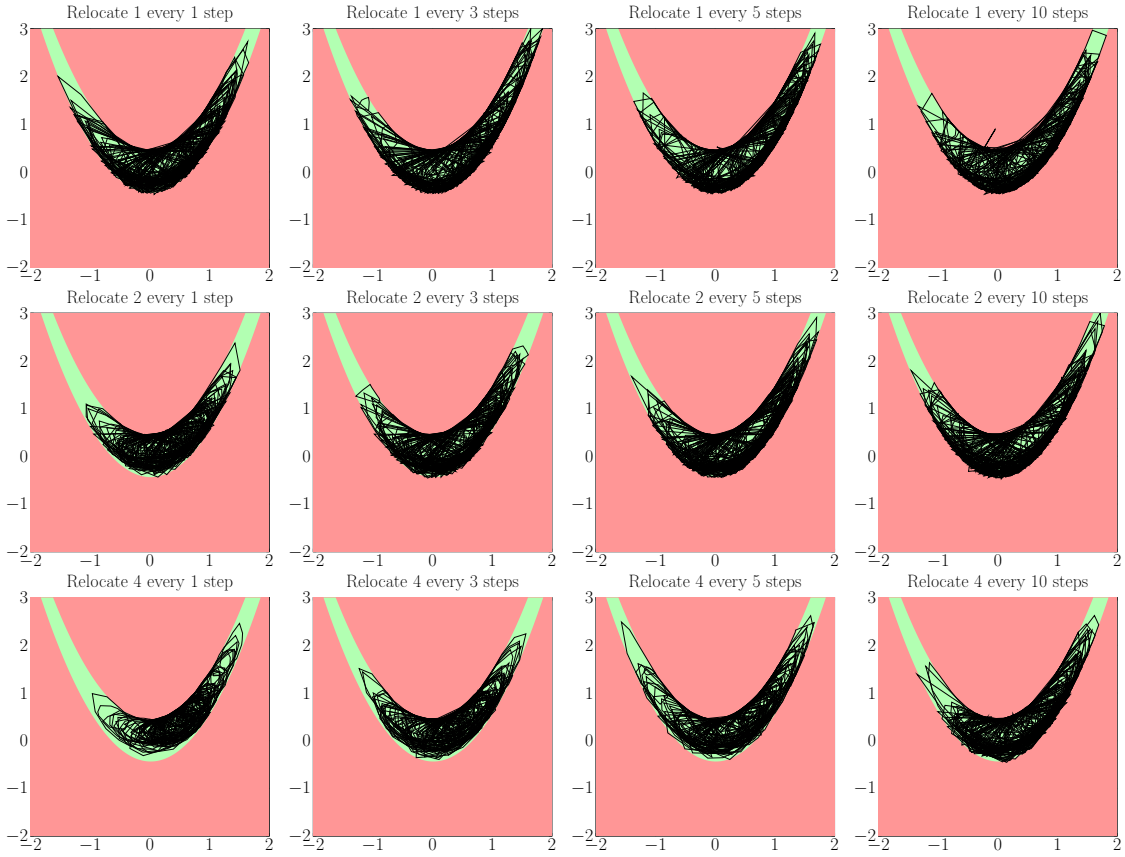

Figure 4.39: 100 polytopes calculated with the "Move Vertex" strategy and different configurations of the "Relocate Vertices" step.

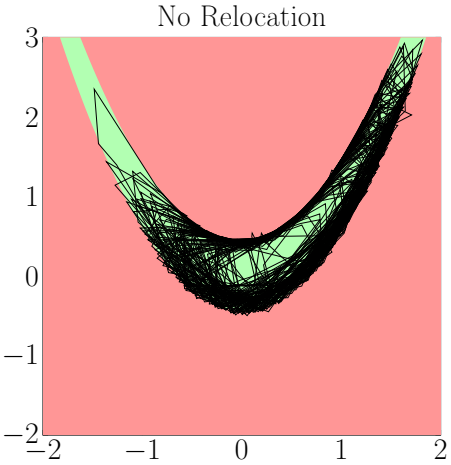

Figure 4.40: 100 polytopes calculated with the "Move Edge" strategy and no vertex relocation.

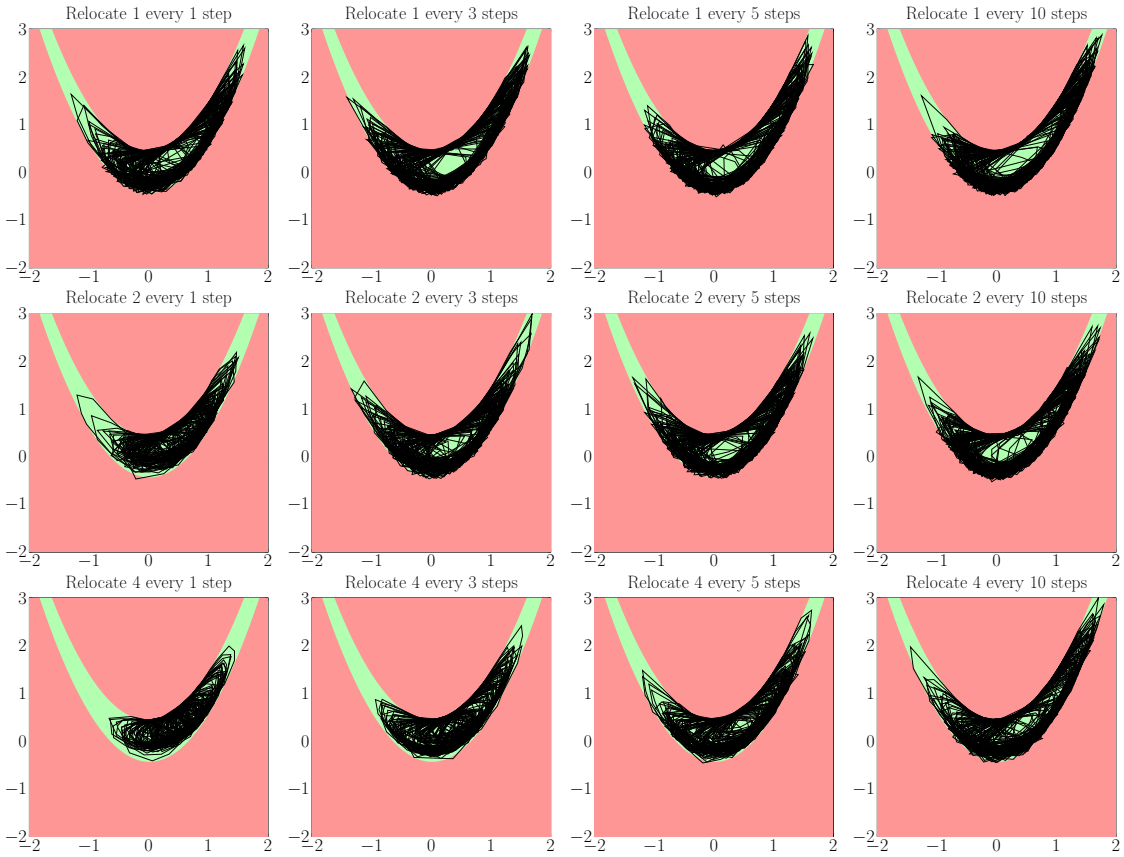

Figure 4.41: 100 polytopes calculated with the “Move Edge” strategy and different configurations of the “Relocate Vertices” step.

Vertex Relocation for 40 Vertices

Finally, we test the vertex relocation on polytopes with 40 vertices. Because we increase the number of vertices, we can also increase the number of vertices which are relocated in each step. Therefore, every 1, 3, 5 or 10 steps, 0, 1, 3, 5 or 10 vertices are relocated. Additionally, vertices are also relocated in the consolidation phase. The results are listed in Table 4.7. As before, the relocation has little influence on the volume of the polytope. The only exception is again when we relocate 10 vertices in every step, where the volume is significantly smaller. This indicates that too many vertices were relocated too often.

When we previously have analyzed the impact of the number of vertices on the performance of the algorithm, we have hypothesized that a more aggressive relocation of vertices could remove branches that develop in the polytope (see Figure 4.28). In Figure 4.42, we plotted 100 polytopes obtained without any vertex relocation and observe that the polytopes develop spikes and branches. Then, in Figure 4.43, we plotted the different configurations of the vertex relocation for 100 polytopes. We can observe that, when we relocate vertices more often, the polytopes develop much less spikes and branches that stretch into the bad design space than without vertex relocation. Also, when we relocate only one vertex every ten steps, this already has a visually significant impact on the number of branches and spikes in a polytope. Thus, for a high number of vertices, we can confirm our hypothesis from before.

From the first row of Figure 4.43, we can conclude that relocating in every step significantly reduces the number of branches a polytope develops, while small spikes can still develop. Conversely, from the last column, we can conclude that increasing the number of vertices that have to be relocated seems to reduce the number of spikes but small branches can still develop. From the picture at the bottom left, we can deduce why the polytopes lost a significant amount of volume when 10 vertices are relocated in every step. The vertex relocation is so strong that polytopes cannot form vertices that would allow them to reach into the uppermost part of the U-shape.

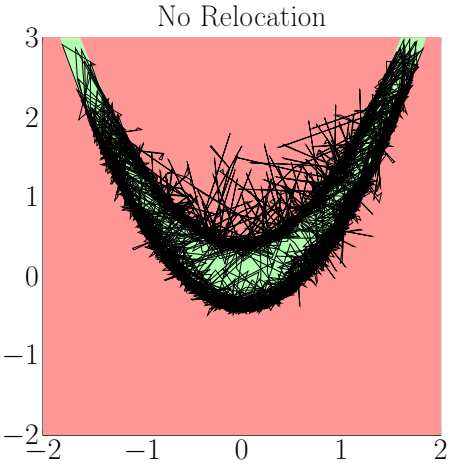

Figure 4.42: The polytope optimization algorithm without vertex relocation for polytopes with 40 vertices.

Relocate k Vertices	Relocate Every ℓ-th Step	Mean	Standard Deviation
$k = 0$	–	0.0919	0.0119
$k = 1$	$\ell = 1$	0.0936	0.0155
	$\ell = 3$	0.0920	0.0127
	$\ell = 5$	0.0927	0.0124
	$\boldsymbol{\ell = 10}$	**0.0945**	**0.0097**
$k = 3$	$\boldsymbol{\ell = 1}$	**0.0954**	**0.0137**
	$\ell = 3$	0.0924	0.0139
	$\ell = 5$	0.0920	0.0158
	$\ell = 10$	0.0936	0.0146
$k = 5$	$\ell = 1$	0.0917	0.0126
	$\boldsymbol{\ell = 3}$	**0.0966**	**0.0115**
	$\ell = 5$	0.0940	0.0131
	$\ell = 10$	0.0963	0.0113
$k = 10$	$\ell = 1$	0.0665	0.0168
	$\ell = 3$	0.0928	0.0130
	$\boldsymbol{\ell = 5}$	**0.0948**	**0.0127**
	$\ell = 10$	0.0942	0.0138

Table 4.7: Mean and standard deviation of the normalized volume for 100 polytopes with 40 vertices and multiple settings for “Relocate Vertices”.

Figure 4.43: Relocating vertices for polytopes with 40 vertices.

Chapter 5

Further Modifications of the Algorithms

This chapter reviews two modifications for all three algorithms that have the potential to increase the volume of the output solution spaces. The first modification swaps the order in which the good and the bad designs are trimmed from a solution space. The second modification analyzes the covariance of an initial set of design points and derives from that a set of 2D-maps $\Omega_{i,j}$. It holds for this set that the pairs of design variables which are coupled most strongly with each other are also part of the same 2D-map. The advantage of these methods is that they are inexpensive to implement and execute. Especially, they can be implemented such that they do not require additional evaluations of the objective function f.

In the following, we will give an account of both modifications, apply them to the box optimization algorithm, the rotated box optimization algorithm and the polytope optimization algorithm and compare the results.

5.1 Swapping Order of Iterations

With the present modification, we change how solution spaces are trimmed. As explained in Subsection 2.1.2, the box optimization algorithm and the rotated box optimization algorithm trim a box by choosing a good design that should remain within the box, and then remove all bad designs. Thus, the algorithms generate one box for each good point and choose from among those the best one as the final trimmed box. We can reverse this order of iterations in the following way: Fix a bad point that should get removed. Then, as above, for each good point, generate a box that removes the bad point but keeps the good point inside. From these, choose the best box. However, this box likely still contains bad designs that have not been removed. Thus, we repeat this procedure for a bad point inside the new box and keep repeating it until no more bad points are inside the box. The pseudo-code of the trimming algorithm with the order of iterations reversed is given in Algorithm 8.

The idea of this modification is that it trims solution spaces more carefully than the original algorithm. By iterating over the bad designs first, much more attention is paid to how a single bad design should be removed. Thus, the algorithm becomes

more conservative. In contrast, the original trimming algorithm simply yields a large variety of possible trimmed solution spaces, from which the best is chosen.

Note that the polytope optimization algorithm has this reversed order of iterations already built in, see Subsection 4.3.2. The modification vastly improved the performance of the polytope optimization algorithm, which is why we decided to use it as the basic setting for the algorithm.

Algorithm 8 (Reversed Order of Iterations). The trimming algorithm from Subsection 2.1.2 with a reversed order of iterations.

Input: Ω_{box}, $\mathcal{X}^{\text{good}}, \mathcal{X}^{\text{bad}}$
Output: $\Omega^{\star}_{\text{box}}$

$\Omega_{\boldsymbol{x}^{\text{bad}}} \leftarrow \Omega_{\text{box}}$
for all $\boldsymbol{x}^{\text{bad}} \in \mathcal{X}^{\text{bad}}$ **do**
  if $\boldsymbol{x}^{\text{bad}} \in \Omega_{\boldsymbol{x}^{\text{bad}}}$ **then**
    for all $\boldsymbol{x}^{\text{good}} \in \mathcal{X}^{\text{good}}$ **do**
      if $\boldsymbol{x}^{\text{good}} \in \Omega_{\boldsymbol{x}^{\text{bad}}}$ **then**
        $\Omega^{\star}_{\boldsymbol{x}^{\text{good}}} \leftarrow \Omega_{\boldsymbol{x}^{\text{bad}}}$
        $\prod_{i=1}^{d} [a_i^{\star}, b_i^{\star}] \leftarrow \Omega^{\star}_{\boldsymbol{x}^{\text{good}}}$
        $[\boldsymbol{n}^{\text{good}}, \boldsymbol{n}^{\text{bad}}] \leftarrow \texttt{countpoints}\big(\boldsymbol{x}^{\text{good}}, \boldsymbol{x}^{\text{bad}}, \Omega^{\star}_{\boldsymbol{x}^{\text{good}}}, \mathcal{X}^{\text{good}}, \mathcal{X}^{\text{bad}}\big)$
        $I_{\text{good}} \leftarrow \left\{ i \in \{1, \dots, d\} \;\middle|\; n_i^{\text{good}} = \min_{j \in \{1,\dots,d\}} n_j^{\text{good}} \right\}$
        $I_{\text{bad}} \leftarrow \left\{ i \in I_{\text{good}} \;\middle|\; n_i^{\text{bad}} = \max_{j \in I_{\text{good}}} n_j^{\text{bad}} \right\}$
        $i^{\star} \in_{\text{rand}} I_{\text{bad}}$
        if $x_{i^{\star}}^{\text{bad}} < x_{i^{\star}}^{\text{good}}$ **then** $a_{i^{\star}}^{\star} \leftarrow x_{i^{\star}}^{\text{bad}}$ **else** $b_{i^{\star}}^{\star} \leftarrow x_{i^{\star}}^{\text{bad}}$ **end if**
        $\mathcal{X}_{\cap}^{\text{good}} \leftarrow \mathcal{X}^{\text{good}} \cap \prod_{i=1}^{d} [a_i^{\star}, b_i^{\star}]$
        for all $a_i^{\star} \neq a_i$ **do** $a_i^{\star} \leftarrow \min_{\boldsymbol{x}^{\text{good}} \in \mathcal{X}_{\cap}^{\text{good}}} x_i^{\text{good}}$ **end for**
        for all $b_i^{\star} \neq b_i$ **do** $b_i^{\star} \leftarrow \max_{\boldsymbol{x}^{\text{good}} \in \mathcal{X}_{\cap}^{\text{good}}} x_i^{\text{good}}$ **end for**
      end if
    end for
    $\Omega_{\boldsymbol{x}^{\text{bad}}} \leftarrow \arg\max_{\Omega^{\star}_{\boldsymbol{x}^{\text{good}}}} \mu\big(\Omega^{\star}_{\boldsymbol{x}^{\text{good}}}\big)$
  end if
end for

Numerical Experiments

We apply each algorithm 100 times to the Rosenbrock function with the settings from Subsections 2.4.1, 3.4.1 and 4.4.1, except that the order of iterations is swapped. The results are listed in Table 5.1 and compared to the results with the standard order of iterations from the corresponding sections. Additionally, we have plotted heat maps for all results in Figure 5.1.

In Figure 5.1, middle row, we can see that the rotated box optimization algorithm seems to move towards the bottom of the U-shape more often if the order of iterations is reversed. However, the impact of the order of iterations on the volume of the box optimization algorithm and the rotated box optimization algorithm is not significant, as the mean normalized volumes and standard deviations for both

algorithms stay about equal. However, the difference is significant for the polytope optimization algorithm. When the bad designs are iterated first, the mean normalized volume is 56% larger than when the good designs are iterated first. Thus, we can justify iterating over the bad designs first as the standard setting for the polytope optimization algorithm.

	Good First		Bad First	
Algorithm	Mean	Standard Deviation	Mean	Standard Deviation
Box Optimization	0.0320	0.0016	0.0317	0.0018
Rotated Box Optimization	0.0332	0.0015	0.0324	0.0029
Polytope Optimization	**0.0529**	**0.0128**	**0.0828**	**0.0178**

Table 5.1: Mean and standard deviation for the normalized volume of 100 polytopes calculated with the two different trimming strategies.The best results are printed in bold

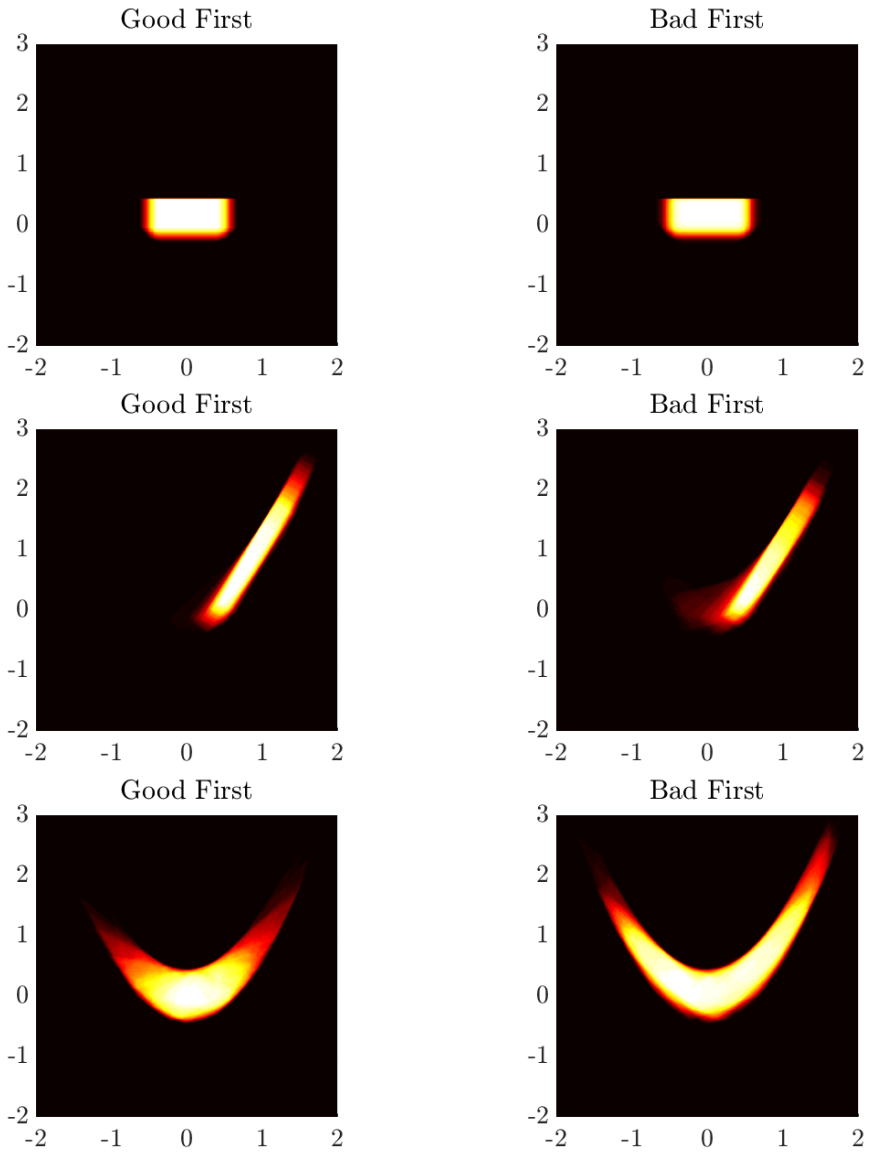

Figure 5.1: Heat maps for the box optimization algorithm (top), the rotated box optimization algorithm (center) and the polytope optimization algorithm (bottom) with different orders of iteration.

5.2 Analysis of Covariance

Sometimes it might not be obvious which pairs of design variables should be chosen for technical reasons. Thus, a way to automatically determine which design variables should be coupled via a 2D-map may be desired.

The idea of this approach is to analyze the covariance matrix of good design points before beginning with the optimization procedure. To this end, design points are randomly sampled from the whole design space Ω_{ds} and all the good design points are put into the matrix

$$\mathbf{X}^{\text{good}} := \left[{}_{(1)}\boldsymbol{x}^{\text{good}}, \ldots, {}_{(n^{\text{good}})}\boldsymbol{x}^{\text{good}} \right] \in \mathbb{R}^{n^{\text{good}} \times d}.$$

Then, the corresponding normalized covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{d \times d}$ is calculated, with

$$\boldsymbol{\Sigma}_{i,j} := \frac{\operatorname{cov}\left(\mathbf{X}_i^{\text{good}}, \mathbf{X}_j^{\text{good}}\right)}{\sigma_i \sigma_j}, \quad i,j = 1, \ldots, d.$$

Here, $\mathbf{X}_i^{\text{good}}$ and $\mathbf{X}_j^{\text{good}}$ denote the i-th and j-th column of the matrix $\mathbf{X}^{\text{good}}$. Their respective sample covariance $\operatorname{cov}\left(\mathbf{X}_i^{\text{good}}, \mathbf{X}_j^{\text{good}}\right)$ is defined by

$$\operatorname{cov}\left(\mathbf{X}_i^{\text{good}}, \mathbf{X}_j^{\text{good}}\right) := \frac{1}{n^{\text{good}} - 1} \sum_{k=1}^{n^{\text{good}}} \left(\mathbf{X}_{i,k}^{\text{good}} - \mu_i\right) \left(\mathbf{X}_{j,k}^{\text{good}} - \mu_j\right),$$

where

$$\mu_i := \frac{1}{n^{\text{good}}} \sum_{k=1}^{n^{\text{good}}} \mathbf{X}_{i,k}^{\text{good}}, \quad \mu_j := \frac{1}{n^{\text{good}}} \sum_{k=1}^{n^{\text{good}}} \mathbf{X}_{j,k}^{\text{good}}$$

is their mean and

$$\sigma_i := \sqrt{\frac{1}{n^{\text{good}} - 1} \sum_{k=1}^{n^{\text{good}}} (\mathbf{X}_{i,k}^{\text{good}} - \mu_i)^2}, \quad \sigma_j := \sqrt{\frac{1}{n^{\text{good}} - 1} \sum_{k=1}^{n^{\text{good}}} (\mathbf{X}_{j,k}^{\text{good}} - \mu_j)^2}$$

is their sample standard deviation. Each entry $\boldsymbol{\Sigma}_{i,j} \in [-1, 1]$ is a so-called *Pearson correlation coefficient* (see [69]) and measures how strongly $\mathbf{X}_i^{\text{good}}$ and $\mathbf{X}_j^{\text{good}}$ are linearly dependent on each other. For example, if $\boldsymbol{\Sigma}_{i,j} = \pm 1$, there exist $a, b \in \mathbb{R}$ with $a > 0$, such that $\mathbf{X}_j^{\text{good}} = \pm a \mathbf{X}_i^{\text{good}} + b$. On the other hand, if $\boldsymbol{\Sigma}_{i,j} \approx 0$, no linear dependence can be inferred, however, $\mathbf{X}_i^{\text{good}}$ and $\mathbf{X}_j^{\text{good}}$ may still be nonlinearly dependent on each other.

Since the covariance matrix $\boldsymbol{\Sigma}$ indicates the dimensions that admit a strong linear coupling, it should be useful for determining the choice of design variables for a 2D-map. The 2D-maps $\Omega_{i,j}$ with the strongest linear coupling of the dimensions i and j can be found iteratively by choosing those pairs (i, j) with $\boldsymbol{\Sigma}_{i,j} = \max_{(k,\ell) \in \mathcal{I}} |\boldsymbol{\Sigma}_{k,\ell}|$ and

$$\mathcal{I} := \left\{ (k, \ell) \in \{1, \ldots, d\}^2 \mid k < \ell \text{ and } k, \ell \text{ not part of another 2D-map} \right\}.$$

It should be noted that this way of determining the coupled pairs of design variables is purely mathematical and does not take aspects of design into account. Furthermore, this method requires additional evaluations of the objective function f. We

need to find at least two good design points within Ω_{ds}. Otherwise, we cannot calculate the covariance matrix. In a high-dimensional design space with a small good design space, we may need a lot of function evaluations to find good design points. If f is costly to evaluate, one may wish to reduce the number of function evaluations done during the optimization algorithms, e.g. by reducing the number of steps in the exploration and the consolidation phase. The function evaluations that are thereby saved could then be used to apply the analysis of covariance.

Application to a 6D Problem from Vehicle Dynamics

Because the analysis of covariance is only useful in a high-dimensional context, we need an appropriate problem to test it. Therefore, we introduce the following 6D problem from vehicle dynamics. It stems from a practical application and is a chassis design problem where some components interact strongly with each other. These interactions give a good natural indication for the choice of 2D-maps. The problem is given by a linear objective function. Thus, the good design space has a polygonal shape, similar to the 2D polygon problem presented in Subsection 2.4.1.

The design space for this problem is $\Omega_{\text{ds}} := [0, 3.5]^2 \times [0.4, 1.6]^2 \times [0.2, 2.3]^2$. Each design variable is a scaling factor of a vehicle characteristic. We refer to Table 5.2 for a short description and the intervals associated with the design variables. It is assumed that the design variables for other vehicle components have already been determined earlier in the design process. The choice of $x_1, \ldots, x_6$ has no influence on those design variables.

Design Variable	Interval	Scaling Factor
x_1	[0,3.5]	force-velocity characteristics of the dampers of the front axle
x_2	[0,3.5]	force-velocity characteristics of the dampers of the rear axle
x_3	[0.4,1.6]	force-displacement of the bearing spring of the front axle
x_4	[0.4,1.6]	force-displacement of the bearing spring of the rear axle
x_5	[0.2,2.3]	stiffness of the anti-roll bar of the front axle
x_6	[0.2,2.3]	stiffness of the anti-roll bar of the rear axle

Table 5.2: Design variables for the 6D problem from vehicle dynamics.

The objective function is derived as follows:

Let

$$\boldsymbol{g}_c := \left[0.421, -0.054, 0.414, 0.724, 0.243, 0.027, 0.371\right]^\top$$

and

$$\mathbf{G} := \begin{bmatrix} 0 & 0 & 0.036 & -0.203 & 0.258 & -0.102 \\ 0 & 0 & 0.111 & -0.313 & 0.540 & -0.165 \\ 0 & 0 & -0.115 & 0.273 & -0.495 & 0.143 \\ -0.152 & -0.025 & -0.078 & 0.121 & -0.329 & 0.057 \\ -0.020 & 0.216 & -0.148 & 0.339 & -0.586 & 0.155 \\ 0 & 0 & 0.088 & 0.144 & 0.391 & 0.072 \\ 0 & 0 & -0.088 & -0.144 & -0.391 & -0.072 \end{bmatrix}.$$

For a design point $\boldsymbol{x} \in \Omega_{\text{ds}}$, normalize it with respect to the cube $[-1,1]^6$ by

$$x_i^{\text{norm}} := 2\frac{x_i - x_i^{\text{low}}}{x_i^{\text{up}} - x_i^{\text{low}}} - 1, \quad i = 1, \ldots, 6,$$

where $\boldsymbol{x}^{\text{low}} := [0.5, 0.5, 0.6, 0.6, 0.5, 0.5]^\top$ and $\boldsymbol{x}^{\text{up}} := [3, 3, 1.4, 1.4, 2, 2]^\top$. Then, apply $\mathbf{G}$ such that

$$\boldsymbol{y} := \mathbf{G}\boldsymbol{x}^{\text{norm}}. \tag{5.2.1}$$

A design is considered to be bad if $y_i \leq g_c^{(i)}$ for all $i = 1, \ldots, 6$. The 2D-maps that are usually coupled for this problem are $C_1 := \Omega_{1,2} \times \Omega_{3,4} \times \Omega_{5,6}$, compare [25]. Finally, this yields the objective function

$$f(\boldsymbol{x}) := \begin{cases} 0, & \text{if } \boldsymbol{x} \in \Omega_{\text{ds}} \text{ and } y_i \leq g_c^{(i)} \text{ for all } i = 1, \ldots, 6, \\ 1, & \text{otherwise,} \end{cases} \tag{5.2.2}$$

where the critical value is $c := 0.5$.

In a first experiment, we test the variability of the analysis of covariance when applied to this problem. The number of sample points used for the analysis strongly influences the kind of 2D-maps that are found. Thus, we apply the analysis of covariance 100 times each for $N = 100$, 200, 300, 500, 1000 and 1500 sample points. As we can see in Figure 5.2, for $N = 100$ sampled design points, the couplings C_1, $C_2 := \Omega_{1,2} \times \Omega_{3,5} \times \Omega_{4,6}$, $C_3 := \Omega_{1,3} \times \Omega_{2,4} \times \Omega_{5,6}$, $C_4 := \Omega_{1,3} \times \Omega_{2,5} \times \Omega_{4,6}$ and $C_5 := \Omega_{1,4} \times \Omega_{2,5} \times \Omega_{3,6}$ are found. C_2, C_4 and C_5 have an approximately equal chance to be selected. The analysis rarely finds the couplings C_1 and C_3. The more the number of sampled designs is increased, the more often the coupling C_2 is found, indicating that for these 2D-maps, the designs display the strongest linear coupling.

Thus, the analysis of covariance may be rather unreliable for a low number of sampled designs. However, we would need only 500 design points (this are only 5% of all designs sampled during the exploration phase in our previous experiments) to produce a more reliable prediction for the strongest coupling. Note that we might require more design points when we apply the analysis of covariance to higher-dimensional problems or problems with a very small good design space.

In a second experiment, we are interested in the quality of all couplings found previously. For $N = 100$, the standard coupling that is applied in [25] (namely $C_1 = \Omega_{1,2} \times \Omega_{3,4} \times \Omega_{5,6}$) is almost never found. Thus, the question arises if all other couplings found for $N = 100$ are better than C_1.

We compare the volume of the solution spaces found by the box optimization algorithm, the rotated box optimization algorithm and the polytope optimization algorithm when applied to the couplings found by the analysis of covariance for $N = 100$

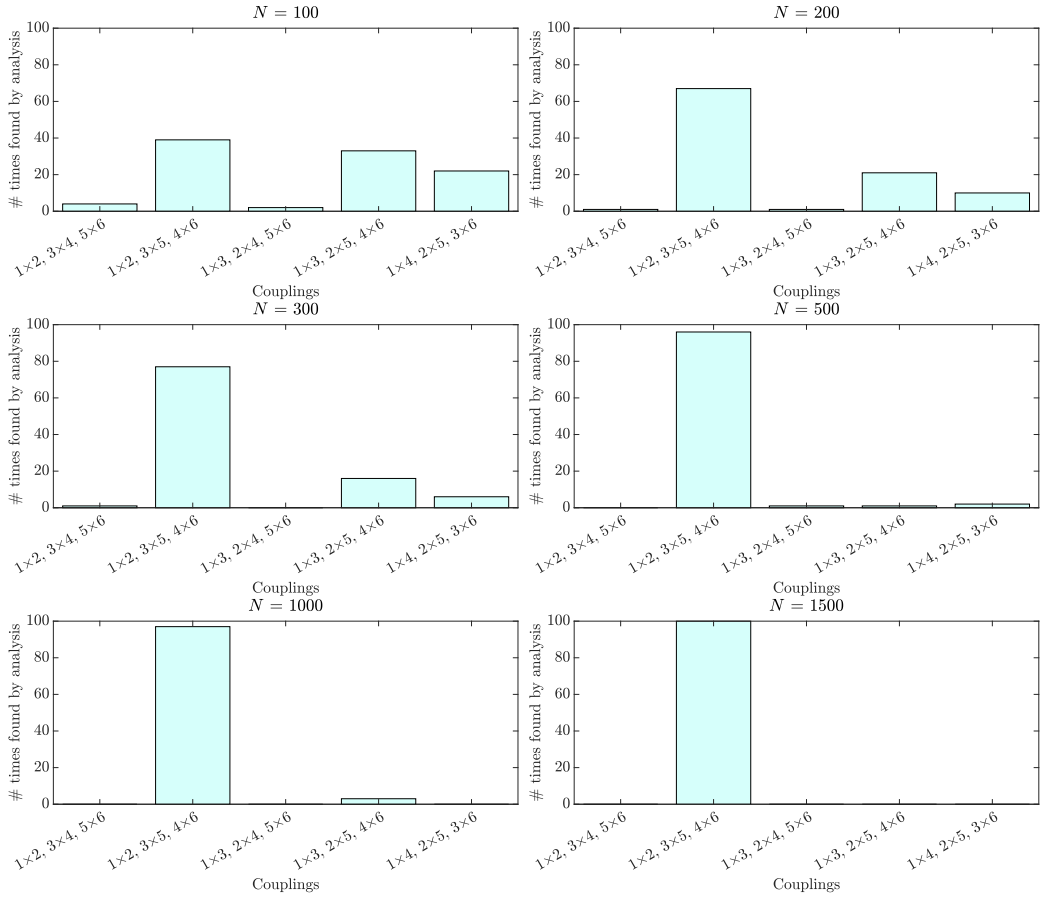

Figure 5.2: The bar plots show how many times the 2D-maps with dimensions coupled as indicated are found by the analysis of covariance.

design points, i.e., the couplings $C_1 = \Omega_{1,2} \times \Omega_{3,4} \times \Omega_{5,6}$, $C_2 = \Omega_{1,2} \times \Omega_{3,5} \times \Omega_{4,6}$, $C_3 = \Omega_{1,3} \times \Omega_{2,5} \times \Omega_{4,6}$, $C_4 = \Omega_{1,4} \times \Omega_{2,5} \times \Omega_{3,6}$ and $C_5 = \Omega_{1,3} \times \Omega_{2,4} \times \Omega_{5,6}$. Additionally, for the purpose of comparing the polytope optimization algorithm with the box optimization algorithm and the rotated box optimization algorithm, any design that lies outside the box $\Omega_{\text{ds}} := [0.5, 3]^2 \times [0.6, 1.4]^2 \times [0.5, 2]^2$ is classified as bad. We do so because the rotated box optimization algorithm treats the boundary of a design space different from the polytope optimization algorithm (see Section 2.2 and Subsection 4.2.8). The artificial boundary of bad points ensures that the results of both algorithms stay comparable. This results in the new objective function

$$f(\boldsymbol{x}) := \begin{cases} 0, & \text{if } \boldsymbol{x} \in \Omega_{\text{ds}} \text{ and } y_i \leq g_c^{(i)} \text{ for all } i = 1, \ldots, 6, \\ 1, & \text{otherwise,} \end{cases}$$

where the critical value is again $c := 0.5$.

For the box optimization algorithm and the rotated box optimization algorithm, we use the same parameter settings as in Subsection 2.4.1 and 3.4.1, except that the initial box is

$$\Omega_{\text{box}} := [1.4, 1.6] \times [0.9, 1.1] \times [0.9, 1.1] \times [0.8, 1] \times [0.8, 1] \times [1.1, 1.3].$$

For the polytope optimization algorithm, we use parameter settings found by our results from Subsection 4.4.2, i.e., $n^{\text{exp}} = n^{\text{con}} = 100$, $N = 100$ design points,

dynamic growth rate with $a^{\text{target}} = 0.6$ and $g^{(0)} = 0.05$, $M = 10$ vertices, required minimum size of angles $\alpha = 5°$, vertex relocation takes place in every step of the exploration phase, relocating the shortest edge, and the trimming strategy is "Edge Walking". The initial polytope is dependent on the chosen combination of 2D-maps. For a vector of polygon centers $\boldsymbol{z} := (1.5, 1, 1, 0.9, 0.9, 1.2)^\top$ and the 2D-maps $\Omega_{i,j}$, the polygon $P_{i,j}$ is given by the vertices

$$\boldsymbol{v}_{i,j}^{(k)} := \begin{pmatrix} z_i + 0.1 \cdot \cos(k \cdot 2\pi/M + \pi/4) \\ z_j + 0.1 \cdot \sin(k \cdot 2\pi/M + \pi/4) \end{pmatrix}$$

for $k = 1, \ldots, M$. The resulting polytope Ω_{pol}, which is the product of all polygons $P_{i,j}$, is comparable in size and position to the initial box Ω_{box} for the box optimization algorithm and the rotated box optimization algorithm. Finally, in [25], a fourth algorithm, called *linear solution space algorithm*, has been applied to this problem for the coupling C_1. It uses an interior-point method (compare [61]) to calculate the largest possible solution space when the objective function is linear, which has been used as a reference solution. Note that [61] considers the objective function in the form of equation (5.2.1), as the function in (5.2.2) is not linear.

The results of the experiment are listed in Table 5.3 and heat maps for the corresponding results are displayed in Figures 5.3, 5.4 and 5.5. Figures 5.6, 5.7 and 5.8 show solution spaces and their surroundings for each optimization algorithm and coupling that have a volume close to the mean. According to these Figures, the solution spaces from the rotated box optimization algorithm and the polytope optimization algorithm fill out most of the good design space and cannot gain much more, which suggests that the results are optimal.

For the standard coupling C_1, the rotated box optimization algorithm finds solution spaces that have 459% of the volume of those calculated by the box optimization algorithm and 50% of the volume of those calculated by the linear solution space algorithm. The polytope optimization algorithm yields solution spaces with 574% of the volume of those calculated by the box optimization algorithm, 125% of the volume of those calculated by the rotated box optimization algorithm and 63% of the volume of those calculated by the linear solution space algorithm. Additionally, in Figure 5.9, we plotted the solution spaces that are close to the mean such that they cover the solution space found by the linear solution space algorithm.

Apparently, the coupling C_2 yields the solution spaces with the largest volume for both, the rotated box optimization algorithm and the polytope optimization algorithm. With this coupling, the volume is more than two times larger than with C_1. However, all other couplings are worse than C_1. They yield solutions spaces with smaller volume, sometimes only half as much as for C_1. The reason for this becomes apparent in Figures 5.4 and 5.5. For the 2D-maps $\Omega_{2,4}$ and $\Omega_{2,5}$, the algorithms are only able to find a small amount of good design space. Hence, a strong linear coupling of two dimensions does not mean that the resulting solution spaces have a large volume. On the other hand, all solution spaces found by the rotated box optimization algorithm and the polytope optimization algorithm have a larger volume than the solution space found by the box optimization. We conclude that any coupling found by the analysis of covariance is better than no coupling, suggesting that it is reasonable to apply the analysis when no prior information about good couplings is available. Finally, it may be worth to apply the analysis even if a coupling is already given, as it might yield an even better coupling.

Algorithm	Coupling	Mean	Standard Deviation
Linear Solution Space	C_1	$3.21 \cdot 10^{-4}$	–
Box Optimization	–	$0.3522 \cdot 10^{-4}$	$0.0987 \cdot 10^{-4}$
Rotated Box Optimization	C_1	$1.6167 \cdot 10^{-4}$	$0.5410 \cdot 10^{-4}$
	$\boldsymbol{C_2}$	$\mathbf{3.5760 \cdot 10^{-4}}$	$\mathbf{1.1942 \cdot 10^{-4}}$
	C_3	$0.8663 \cdot 10^{-4}$	$0.3918 \cdot 10^{-4}$
	C_4	$0.6774 \cdot 10^{-4}$	$0.3008 \cdot 10^{-4}$
	C_5	$0.7066 \cdot 10^{-4}$	$0.2855 \cdot 10^{-4}$
Polytope Optimization	C_1	$2.0232 \cdot 10^{-4}$	$0.4402 \cdot 10^{-4}$
	$\boldsymbol{C_2}$	$\mathbf{4.0857 \cdot 10^{-4}}$	$\mathbf{0.7613 \cdot 10^{-4}}$
	C_3	$1.8035 \cdot 10^{-4}$	$0.4186 \cdot 10^{-4}$
	C_4	$1.0744 \cdot 10^{-4}$	$0.2624 \cdot 10^{-4}$
	C_5	$1.4094 \cdot 10^{-4}$	$0.3304 \cdot 10^{-4}$

Table 5.3: Mean and standard deviation of the normalized volume for 100 solution spaces calculated with different couplings for the box optimization algorithm, the rotated box optimization algorithm and the polytope optimization algorithm. The result of the linear solution space algorithm is tabulated for the sake of completeness. The best couplings are printed in bold.

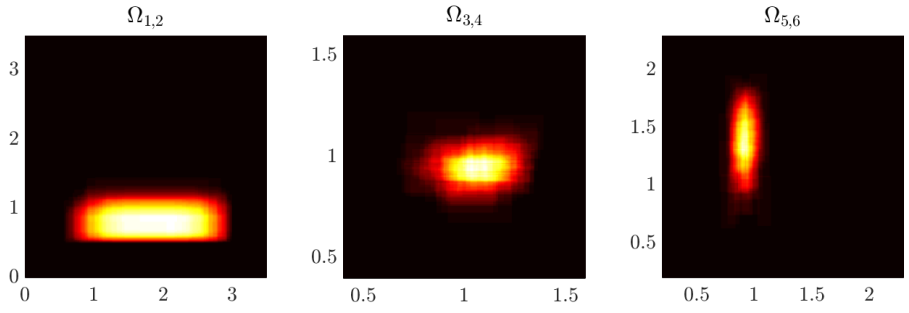

Figure 5.3: Heat maps for the box optimization algorithm, plotted in the 2D-maps for C_1.

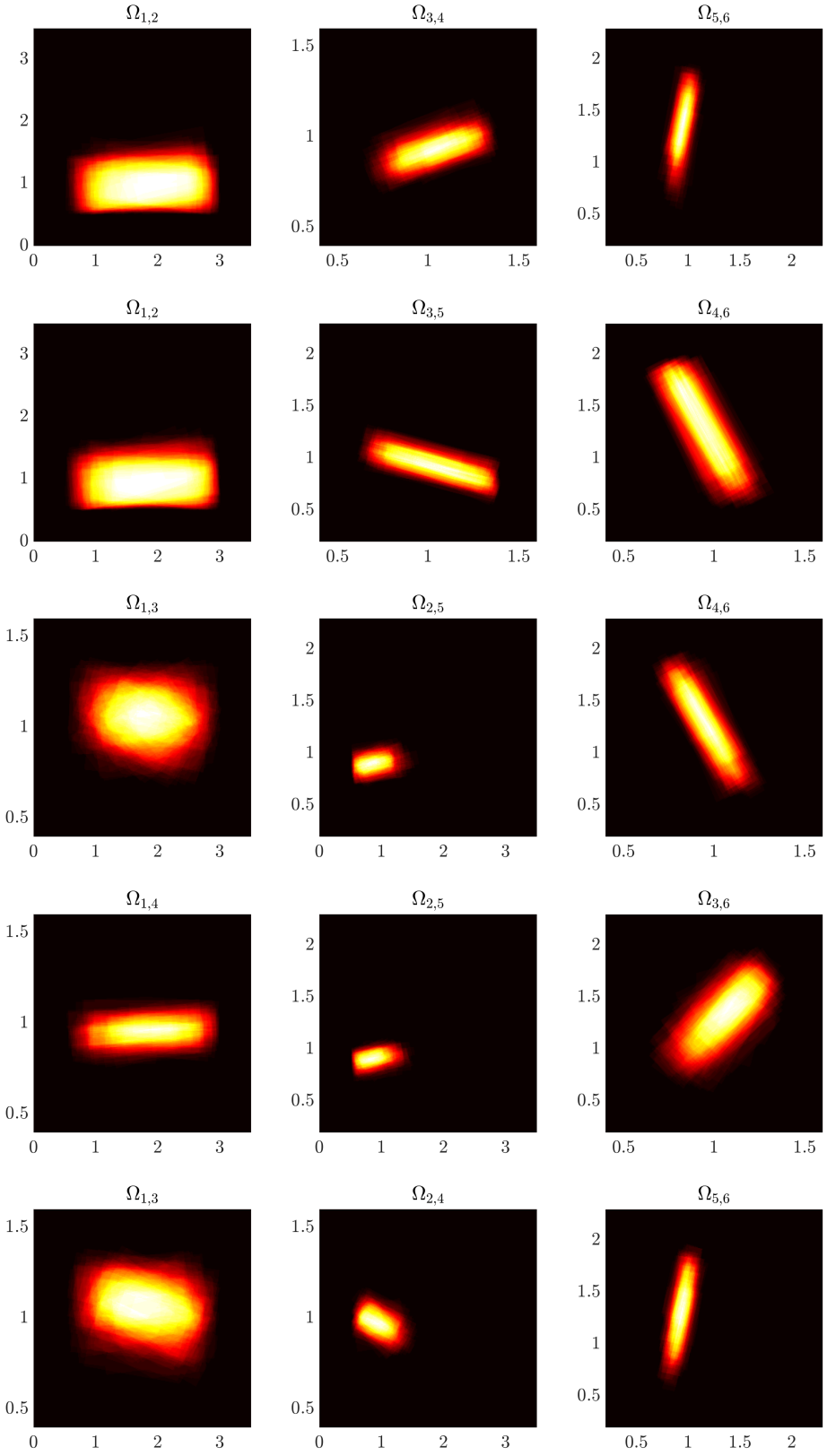

Figure 5.4: From top to bottom: heat maps for the couplings C_1 to C_5, computed by the rotated box optimization algorithm.

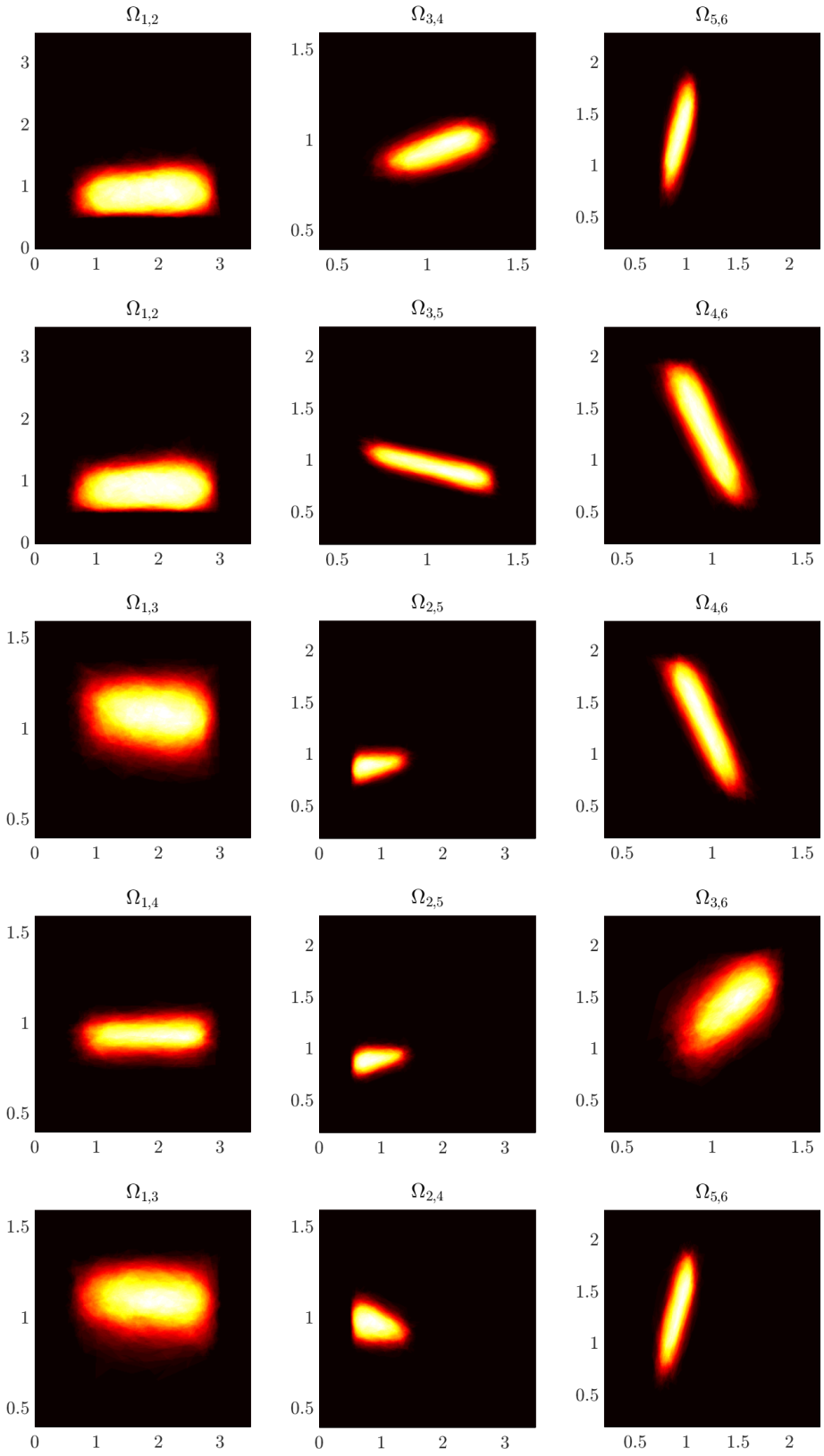

Figure 5.5: From top to bottom: heat maps for the couplings C_1 to C_5, computed by the polytope optimization algorithm.

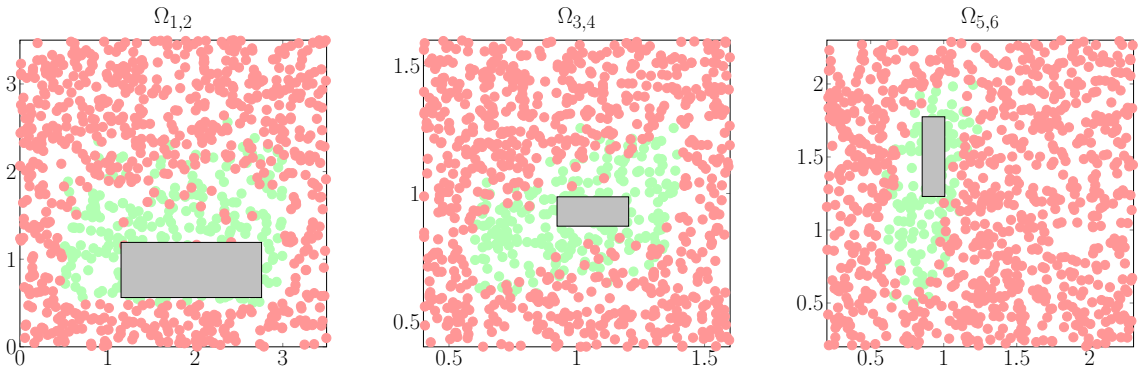

Figure 5.6: A solution space close to the mean normalized volume of all solution spaces found by the box optimization algorithm, plotted in the 2D-maps for C_1.

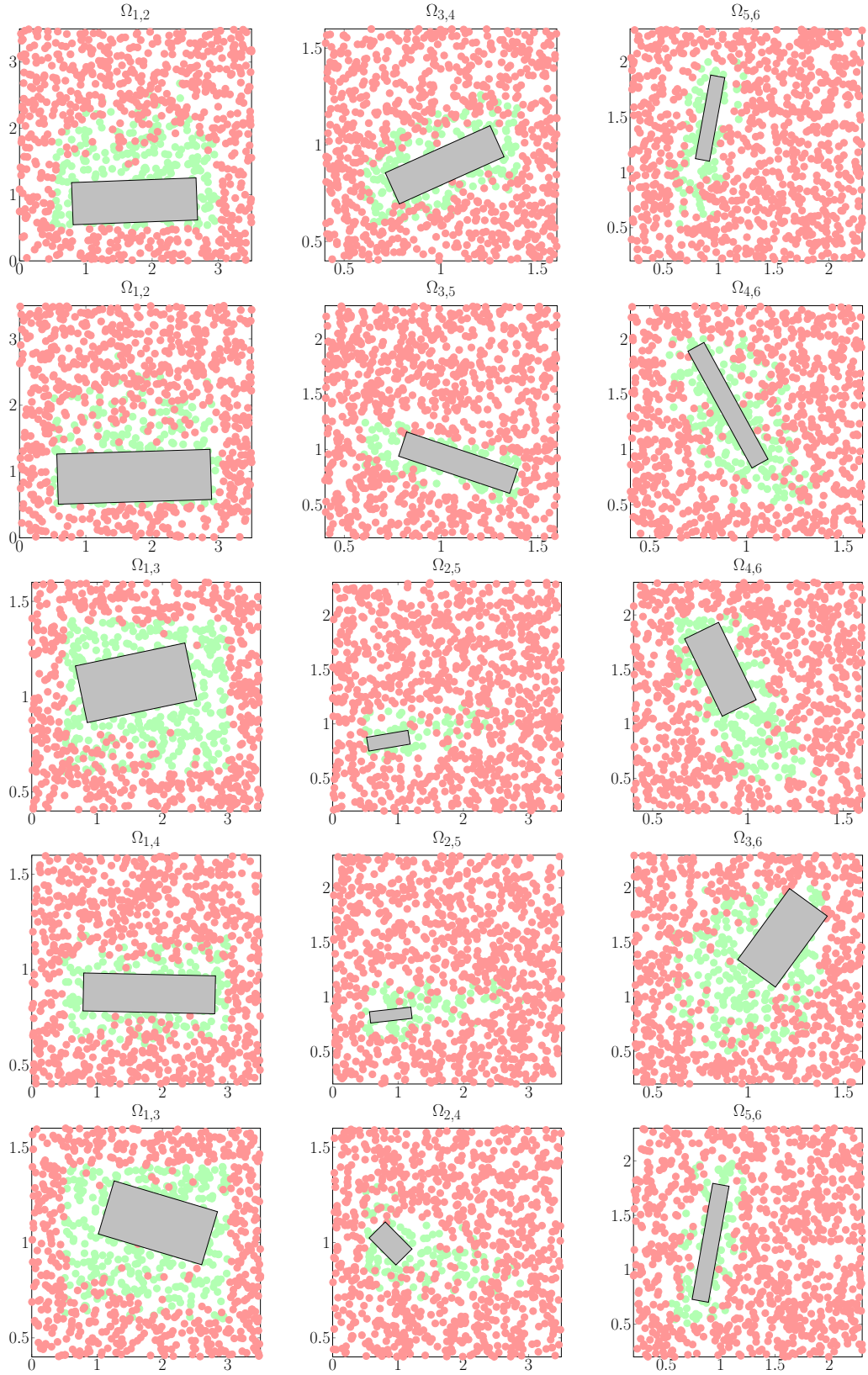

Figure 5.7: From top to bottom: rotated boxes close to the mean normalized volume for the couplings C_1 to C_5.

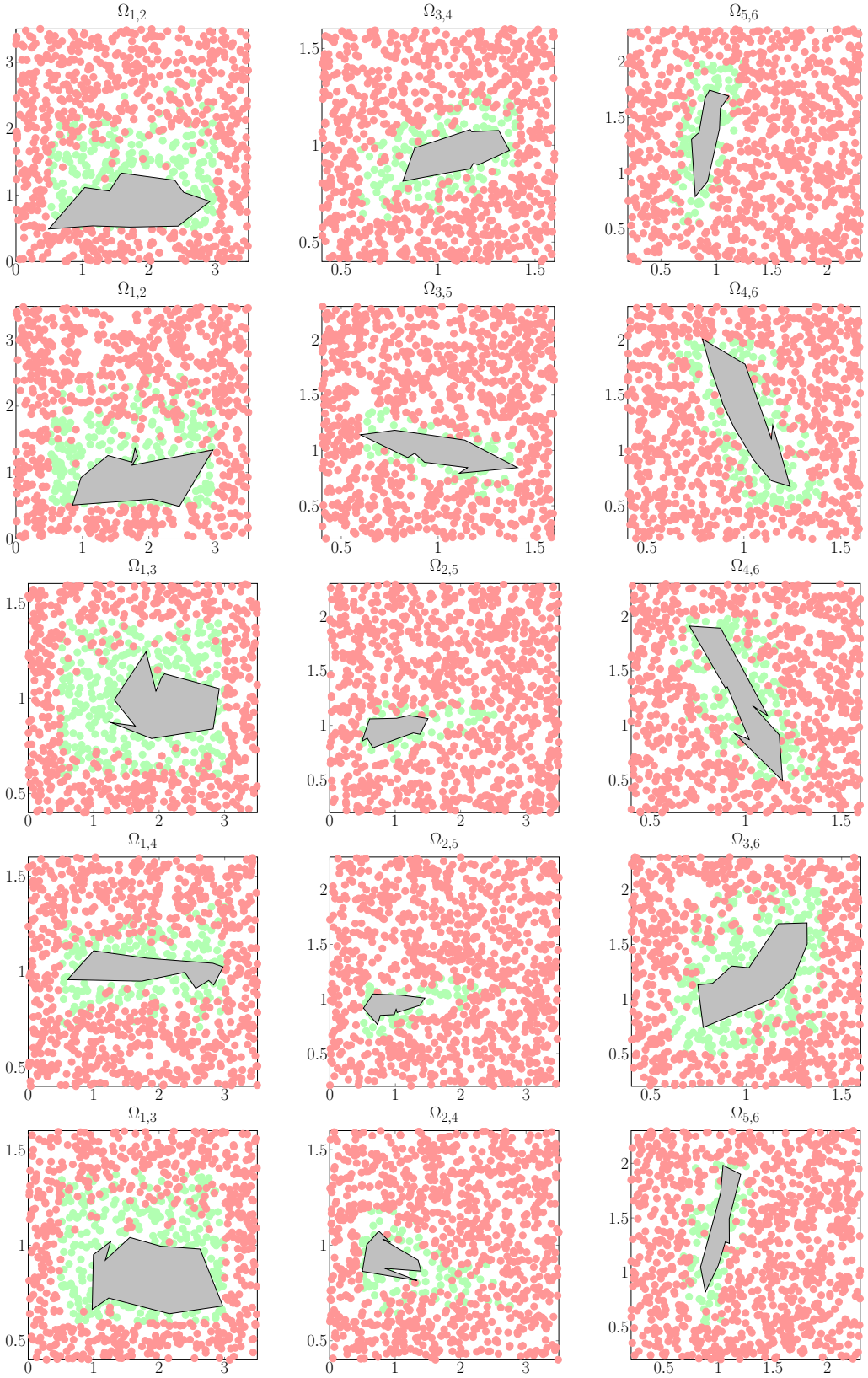

Figure 5.8: From top to bottom: polytopes close to the mean normalized volume for the couplings C_1 to C_5, computed by the polytope optimization algorithm.

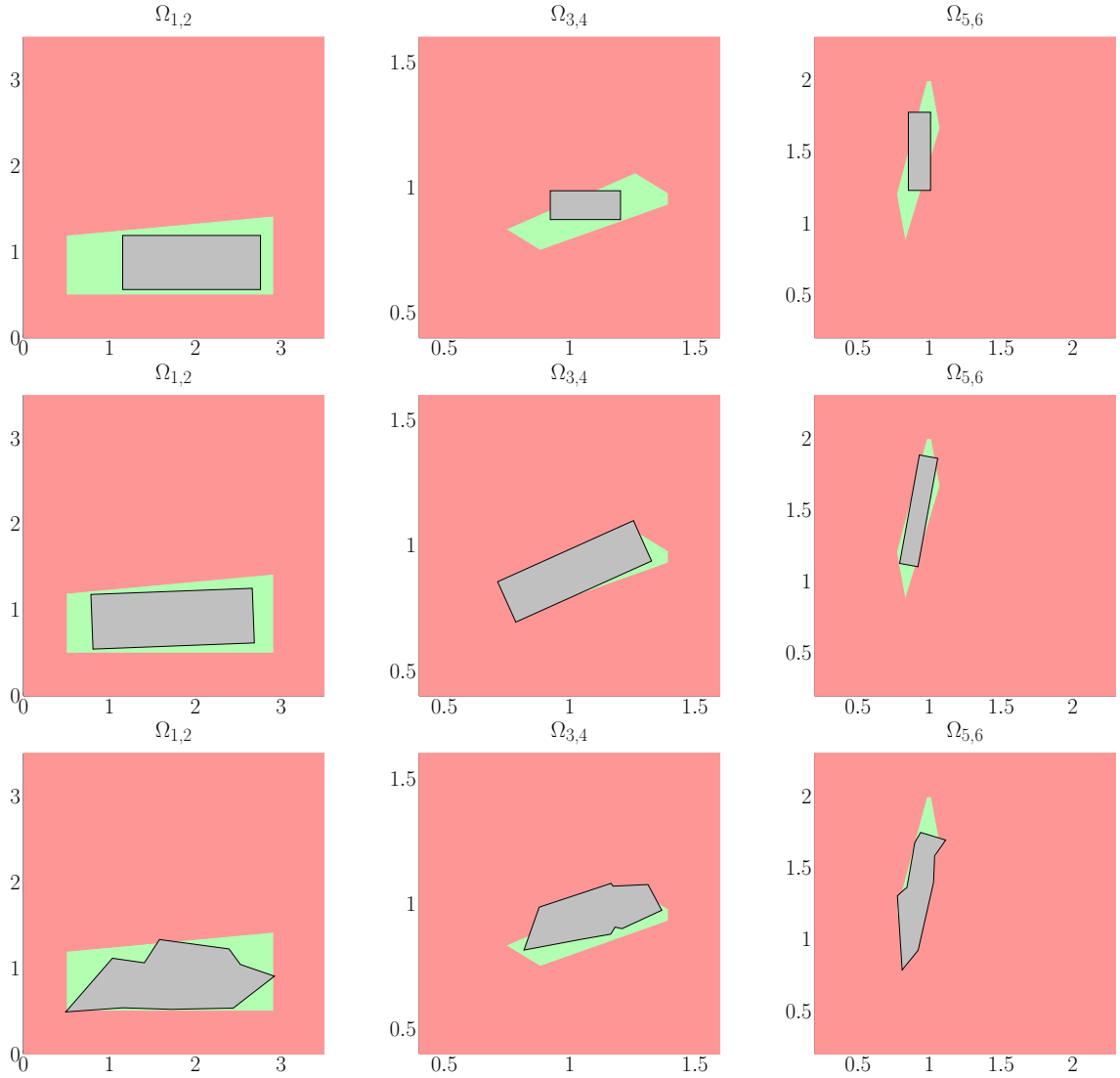

Figure 5.9: From top to bottom: solution spaces close to the mean from each optimization algorithm, laid over over the solution space (in green) found by the linear solution space algorithm.

Chapter 6

Numerical Results

In this chapter, we apply the box optimization algorithm, the rotated box optimization algorithm and the polytope optimization algorithm to three different problems. The problems are a 4D variant of the Rosenbrock function, the 8D nonlinear acoustical engineering problem we already considered in Subsection 2.4.2, and a 10D optimal control problem. The goal of this chapter is to monitor the performance of each optimization algorithm in a higher-dimensional context and compare their results.

The settings for all parameters and optimization algorithms do not change throughout this chapter unless mentioned explicitly. They are similar to settings from previous chapters, i.e., each algorithm is applied 100 times to each optimization problem, $n^{\text{exp}} = n^{\text{con}} = 100$, $N = 100$ design points and the growth rate is dynamic with $a^{\text{target}} = 0.6$ and $g^{(0)} = 0.05$. For the polytope optimization algorithm, each 2D-polygon has $M = 10$ vertices, the required minimum size of angles is $\alpha = 5°$, vertex relocation takes place in every step of the exploration phase, relocating one edge, and the trimming strategy is "Edge Walking".

6.1 4D Rosenbrock Function

It is possible to define the Rosenbrock function in more than two dimensions. For $\boldsymbol{x} \in \Omega_{\text{ds}} := \big([-2,2] \times [-2,3]\big)^{d/2}$, where d is even, it is given by the formula

$$f(\boldsymbol{x}) := \sum_{i=1}^{d/2} (1 - x_{2i-1})^2 + 100\big(x_{2i} - x_{2i-1}^2\big)^2.$$

We apply the box optimization algorithm, rotated box optimization algorithm and the polytope optimization algorithm 100 times to this problem with $d := 4$ and $c := 120$. The 2D-maps for the rotated box optimization algorithm and the polytope optimization algorithm are set to $\Omega_{1,2} := \Omega_{3,4} := [-2,2] \times [-2,3]$.

For the polytope optimization algorithm, the initial polytope is given by $\Omega_{\text{pol}} := P_{1,2} \times P_{3,4}$, where $P_{i,j}$ is the initial polygon on the 2D-map $\Omega_{i,j}$ given by the vertices

$$\boldsymbol{v}_{i,j}^{(k)} := \begin{pmatrix} 1.3 + 0.2 \cdot \cos(k \cdot 2\pi/M + \pi/4) \\ 1.8 + 0.2 \cdot \sin(k \cdot 2\pi/M + \pi/4) \end{pmatrix}$$

for $k = 1, \ldots, M$.

The mean normalized volume of all solution spaces found by the three algorithms is given in Table 6.1. In Figure 6.1, we plotted solution spaces that are close to the mean attained by each algorithm. There, the good design space around each solution space on a 2D-map is U-shaped, similar to the 2D Rosenbrock function. From the heat maps in Figure 6.2, we can gather that the polytope optimization algorithm is able to fill out most of the U-shape, whereas the box optimization algorithm and the rotated box optimization algorithm only find hyperboxes in the bottom or on one side of the U-shape. Subsequently, the solution spaces found by the polytope optimization algorithm have approximately 500% of the volume of those found by the other two algorithms.

Algorithm	Mean Normalized Volume	Standard Deviation
Box Optimization	0.0077	0.0019
Rotated Box Optimization	0.0078	0.0009
Polytope Optimization	**0.0399**	**0.0061**

Table 6.1: Results of the different optimization techniques for the 6D problem. The best result is printed in bold.

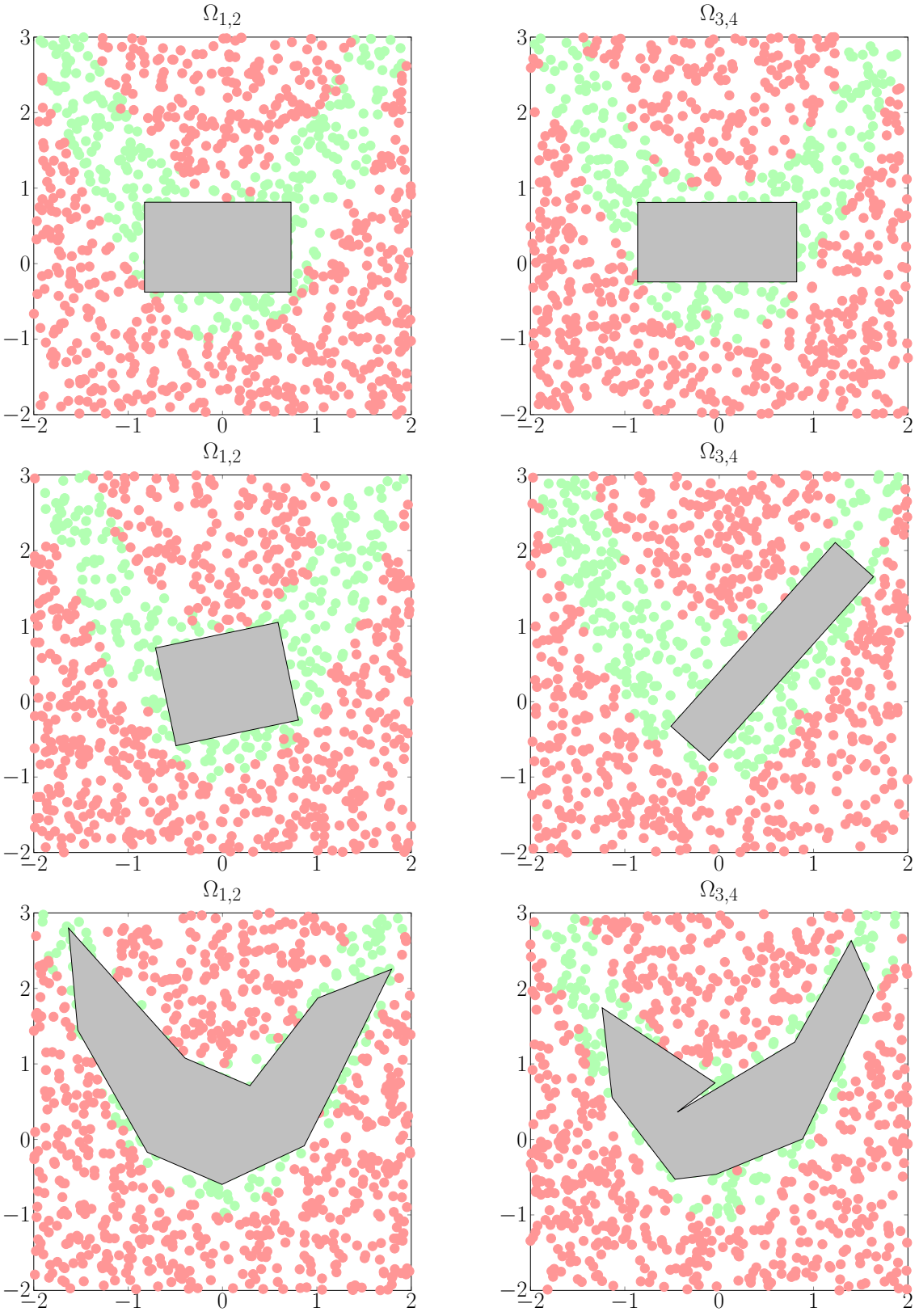

Figure 6.1: Solution spaces with volume close to the mean for the 4D Rosenbrock function.

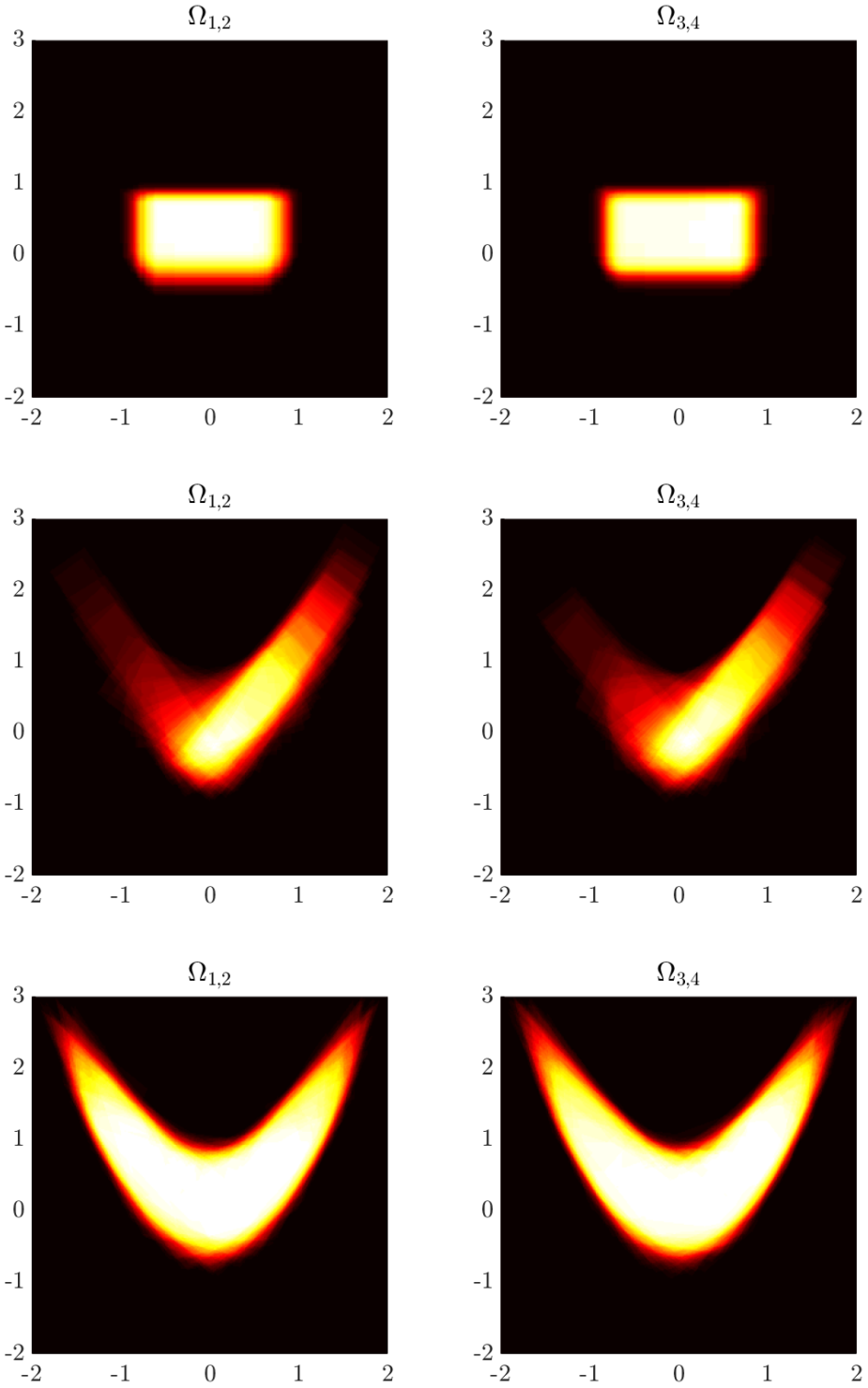

Figure 6.2: Heat maps for the 4D Rosenbrock function.

6.2 8D Nonlinear Problem from Acoustics

Next, we consider the 8D nonlinear engineering problem from acoustics, described previously in Subsection 2.4.2. Recall that the objective function is given by

$$f(\boldsymbol{x}) = \left| 1 + \sum_{\ell=1}^{8} e^{ix_\ell} \right|$$

while $\Omega_{\text{ds}} = [0, 2\pi]^8$ and $c = 1.5$. The rotated box optimization algorithm and the polytope optimization algorithm are coupled on the 2D-maps $\Omega_{1,2} := \Omega_{3,4} := \Omega_{5,6} := \Omega_{7,8} := [0, 2\pi]^2$.

For the box optimization algorithm and the rotated box optimization algorithm, the initial box is given by

$$\Omega_{\text{box}} := \prod_{i=1}^{8} [a_i, b_i]$$

with a_i and b_i as in Table 6.2.

i	1	2	3	4	5	6	7	8
a_i	4.7545	4.9248	1.8186	0.4240	3.2106	1.2389	4.6416	1.2389
b_i	5.6455	5.2958	2.4098	1.1439	3.6663	1.7502	5.0973	1.7502

Table 6.2: Intervals for the initial box of the acoustics problem.

For the polytope optimization algorithm, the initial polytope is $\Omega_{\text{pol}} := P_{1,2} \times P_{3,4} \times P_{5,6} \times P_{7,8}$. It is the product of the 2D polygons $P_{i,j}$ on each 2D-map $\Omega_{i,j}$, where their vertices are given by

$$\boldsymbol{v}_{i,j}^{(k)} := \begin{pmatrix} z_i + 0.4 \cdot \cos(k \cdot 2\pi/M + \pi/4) \\ z_j + 0.4 \cdot \sin(k \cdot 2\pi/M + \pi/4) \end{pmatrix}$$

for $k = 1, \ldots, M$ and by the vector of polygon centers

$$\boldsymbol{z} := (5.2, 5.1, 2.1, 0.8, 3.45, 1.5, 4.9, 1.5)^\top.$$

Algorithm	Mean Normalized Volume	Standard Deviation
Box Optimization	$0.0595 \cdot 10^{-6}$	$0.0307 \cdot 10^{-6}$
Rotated Box Optimization	$1.9659 \cdot 10^{-6}$	$1.9719 \cdot 10^{-6}$
Polytope Optimization	$\mathbf{5.5753 \cdot 10^{-6}}$	$\mathbf{2.2412 \cdot 10^{-6}}$

Table 6.3: Results of the different optimization techniques for the 8D acoustics problem.

The mean normalized volumes of the solution spaces found by the box optimization algorithm, the rotated box optimization algorithm and the polytope optimization algorithm are listed in Table 6.3. Solution spaces from each optimization algorithm

that are close to the mean are plotted in Figure 6.3 and heat maps for each algorithm are shown in Figure 6.4. The rotated box optimization algorithm finds solution spaces with approximately 3300% of the volume of those found by the box optimization algorithm. The polytope optimization algorithm finds solution spaces with approximately 9400% of the volume of those found by the box optimization algorithm and approximately 280% of the volume of those found by the rotated box optimization algorithm. While we find solution spaces with significantly more volume than the box optimization algorithm when we apply either of the rotated box optimization algorithm or the polytope optimization algorithm, the amount of volume we gain by applying the polytope optimization algorithm over the rotated box optimization algorithm is not equally significant. For example, in Figure 6.3, we can see that the rotated box (second row) is similar in size to the polytope (third row) on each 2D-map. Additionally, the heat maps for the rotated box optimization algorithm and the polytope optimization algorithm in Figure 6.4 show that the solution spaces for both algorithms lie in similar regions, although their positions vary more in case of the rotated box optimization algorithm.

However, if we plot the volumes of the solution spaces from the rotated box optimization algorithm and the polytope optimization algorithm, we notice that the volumes from the rotated box optimization algorithm are heavily skewed towards the left-hand side, see Figure 6.5. Recall that the *skewness* of a random variable X with mean μ and variance σ^2 is its third standardized moment (see [8]) and defined as

$$S[X] := E\left[\left(\frac{X-\mu}{\sigma}\right)^3\right].$$

For a random variable with a distribution that is symmetric about its mean, i.e., if $X-\mu$ and $\mu-X$ have the same distribution, it holds $S[X]=0$. If $S[X]>0$, this indicates that the right tail of the distribution is relatively longer or heavier than the left tail, and vice versa if $S[X]<0$. Likewise, the *sample skewness* for a set of scalar data $x_1,\dots,x_n\in\mathbb{R}$ with mean $\bar{x}$ is defined as

$$s := \frac{\frac{1}{n}\sum_{i=1}^n (x_i-\bar{x})^3}{\left(\sqrt{\frac{1}{n}\sum_{i=1}^n (x_i-\bar{x})^2}\right)^3}.$$

If $s>0$, the data right of the mean are more spread out than those left of the mean (as seen in Figure 6.5) and vice versa for $s<0$. The skewness for the volumes found by the rotated box optimization algorithm is 1.6825 and the skewness for those found by the polytope optimization algorithm is 0.4810. Thus, the volumes from the rotated box optimization algorithm are much more skewed and have more outliers than those from the polytope optimization algorithm. This leads to the following question: How are these outliers generated?

To answer this question, we fix the values in five dimensions of the objective function and consider the now three-dimensional objective function

$$f(\boldsymbol{x}) := \left|1+2e^{2i}+e^{3i}+2e^{5i}+\sum_{\ell=1}^{3} e^{ix_\ell}\right|.$$

For this function, we are able to visualize the good and the bad design space, see Figure 6.6. Apparently, the two spaces are heavily intertwined, such that, when

we sample a few design points and project them onto the 2D-map $\Omega_{1,2}$ (see Figure 6.7), there is no region with only bad design space. Instead, the bad design points are scattered among the good design points. This overlapping of the design spaces implies that, whenever we would sample in a region of good designs, it is likely that a few bad design points are sampled, too.

This effect is also visible on the 2D-maps in Figure 6.3. On the 2D-map $\Omega_{1,2}$, the box optimization algorithm and the rotated box optimization algorithm form small boxes inside a region where they could spread out more, thereby losing good design space. However, the polytope optimization algorithm seems to be able to adapt to the effect. When a single bad design point gets sampled in the region of good designs, the algorithm can simply move one edge such that the bad design point gets removed. It is likely to form a spike by doing so (as seen on all the 2D-maps for the polytope optimization algorithm in Figure 6.3), however, most other edges of the polytope stay where they are, such that the polytope keeps most of its volume. Thus, in the case of overlapping good and bad design space, spikes seem to be beneficial for the polytope optimization algorithm, as they allow the algorithm to dodge outliers. This in turn reduces the amount of solution spaces with an outlying volume, which increases the consistency of the algorithm when compared to the rotated box optimization algorithm.

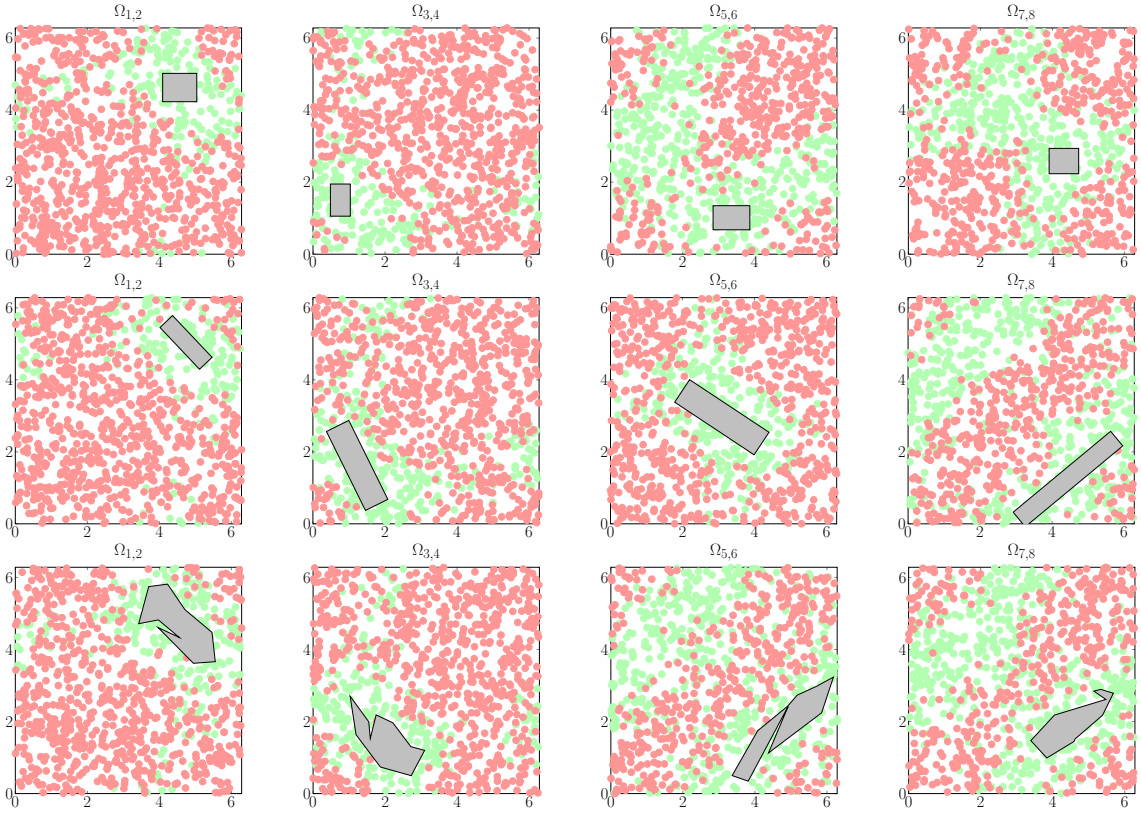

Figure 6.3: Solution spaces with volume close to the mean for the 8D acoustics problem.

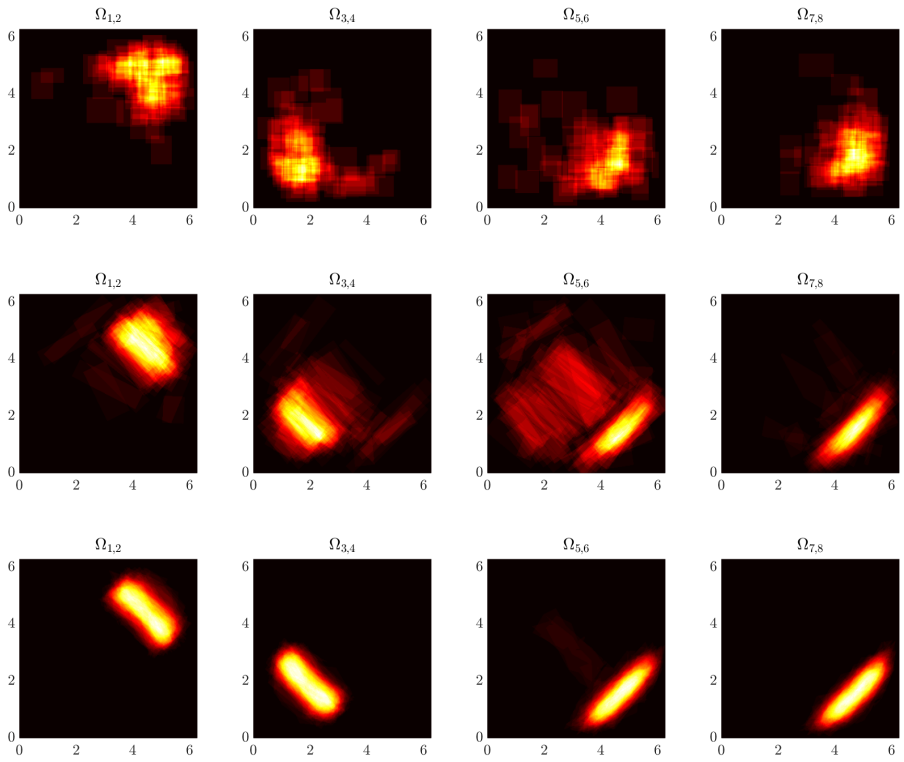

Figure 6.4: Heat maps for the 8D acoustics problem.

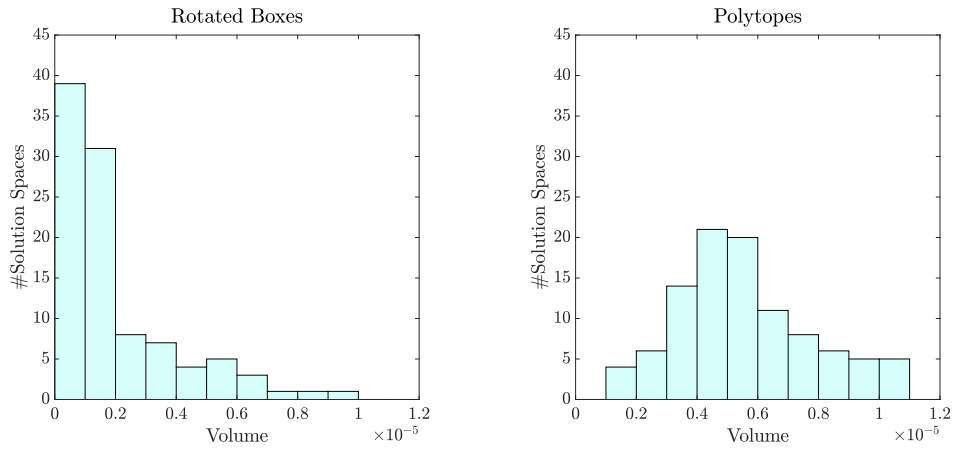

Figure 6.5: Histograms for the rotated box optimization algorithm and the polytope optimization algorithm.

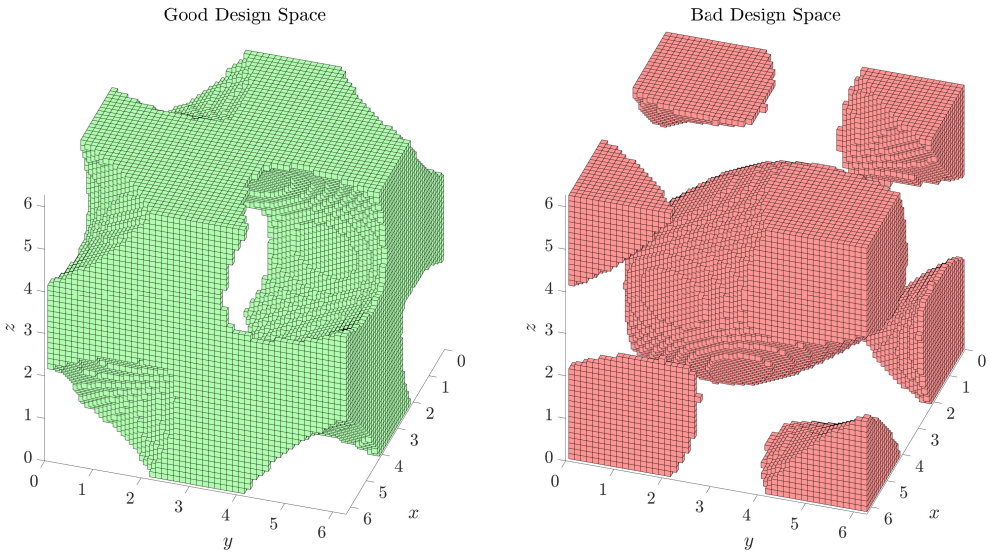

Figure 6.6: Voxel plot of the good (left) and the bad (right) design space for the acoustics problem in 3D.

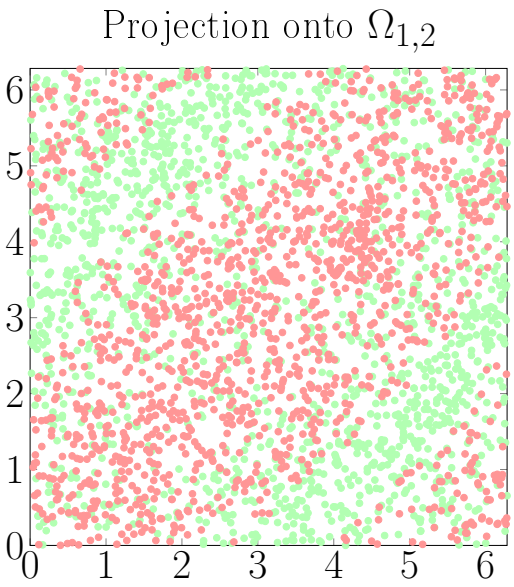

Figure 6.7: Projection of samples points from the 3D acoustics problem onto $\Omega_{1,2}$.

6.3 10D Problem from Optimal Control

We consider the following problem: Five heat sources have to be designed such that they keep the temperature in a control volume on a given constant level. Each heat source has a fixed position $\boldsymbol{x}_i := (x_{i,1}, x_{i,2})^\top$ in the control volume and a circular shape with radius r_i. The temperature at the i-th heat source is given by the constant factor t_i. The distribution of the heat $u(\boldsymbol{x})$ emitted by the heat sources throughout the control volume is modelled by the *steady-state heat equation* (compare [77])

$$\begin{aligned} -\Delta u(\boldsymbol{x}) &= g_{\boldsymbol{r},\boldsymbol{t}}(\boldsymbol{x}), & \boldsymbol{x} &\in D := (0,1)^2, \\ u(\boldsymbol{x}) &= 0, & \boldsymbol{x} &\in \Gamma := \partial D, \end{aligned} \tag{6.3.1}$$

where

$$g_{\boldsymbol{r},\boldsymbol{t}}(\boldsymbol{x}) := \sum_{i=1}^{5} t_i \cdot \mathcal{X}_{B_{r_i}(\boldsymbol{x}_i)}(\boldsymbol{x}).$$

Here, $\mathcal{X}_B$ is the characteristic function

$$\mathcal{X}_B(\boldsymbol{x}) := \begin{cases} 1, & \boldsymbol{x} \in B, \\ 0, & \text{else}, \end{cases}$$

and $B_{r_i}(\boldsymbol{x}_i)$ denotes the ball with radius r_i around the center $\boldsymbol{x}_i$. The positions of the centers $\boldsymbol{x}_i$ are given by

$$\begin{gathered} \boldsymbol{x}_1 := (0.15, 0.4)^\top, \quad \boldsymbol{x}_2 := (0.45, 0.9)^\top, \quad \boldsymbol{x}_3 := (0.87, 0.7)^\top, \\ \boldsymbol{x}_4 := (0.88, 0.25)^\top, \quad \boldsymbol{x}_5 := (0.5, 0.3)^\top. \end{gathered}$$

We intend to determine the variables $\boldsymbol{r}$ and $\boldsymbol{t}$ such that the maximum deviation from the desired temperature, i.e.

$$f(\boldsymbol{r}, \boldsymbol{t}) := \|u^\star_{\boldsymbol{r},\boldsymbol{t}} - u_d\|_{L^\infty(K)},$$

is minimized. Here, $u_d := 0.5$ is the desired constant temperature and $K := [0.3, 0.7]^2 \subset D$ is the region inside the control volume where that temperature should be close to u_d. The problem under consideration is an optimal control problem, where $\boldsymbol{r}$ and $\boldsymbol{t}$ are the *control variables* to be determined such that they minimize the *cost function* f, see [77] for example.

In the context of solution spaces, f is the objective function and $\boldsymbol{r}$ and $\boldsymbol{t}$ are the design variables, where we choose $\Omega_{\text{ds}} := \Omega_{\boldsymbol{r}} \times \Omega_{\boldsymbol{t}}$ as design space with

$$\begin{aligned} \Omega_{\boldsymbol{r}} &:= [0.01, 0.17] \times [0.01, 0.2] \times [0.01, 0.2] \times [0.01, 0.18] \times [0.01, 0.15], \\ \Omega_{\boldsymbol{t}} &:= [0, 100] \times [0, 80] \times [0, 150] \times [0, 180] \times [0, 100]. \end{aligned}$$

The radius and temperature of each heat source are coupled by a 2D-map such that $\Omega_{1,2} := [0.01, 0.17] \times [0, 100]$, $\Omega_{3,4} := [0.01, 0.2] \times [0, 80]$, $\Omega_{5,6} := [0.01, 0.2] \times [0, 150]$, $\Omega_{7,8} := [0.01, 0.18] \times [0, 180]$, and $\Omega_{9,10} := [0.01, 0.15] \times [0, 100]$. As critical value we choose $c := 0.15$, which means that the temperature generated by the heat sources is allowed to deviate up to 30% from the desired temperature.

The results of the algorithms are listed in Table 6.4. Solution spaces close to the mean normalized volume of each algorithm are plotted in Figure 6.8. Similar to the previous problem, the rotated box optimization algorithm and the polytope optimization algorithm yield solution spaces with significantly more volume than those found by the box optimization algorithm. And again, the volume of the solution spaces found by the polytope optimization algorithm is only 279% of the volume of those found by the rotated box optimization algorithm. Additionally, for the rotated box optimization algorithm, the standard deviation is not as large in relation to the mean (only 53%) as in the previous problem. Thus, the rotated box optimization algorithm and the polytope optimization algorithm both yield reasonable results for this problem. However, as can be seen on the heat maps in Figure 6.9, for the 2D-map $\Omega_{9,10}$, the rotated box optimization algorithm struggles to consistently find a suitable region of good design space. In contrast, the polytope optimization algorithm reliably moves into the same region of good design space for every execution of the algorithm.

A solution of (6.3.1) with a design taken from a solution space computed by the polytope optimization algorithm is visualized in Figure 6.10.

Algorithm	Mean Normalized Volume	Standard Deviation
Box Optimization	$0.0045 \cdot 10^{-6}$	$0.0018 \cdot 10^{-6}$
Rotated Box Optimization	$1.1636 \cdot 10^{-6}$	$0.6135 \cdot 10^{-6}$
Polytope Optimization	$\mathbf{3.2493 \cdot 10^{-6}}$	$\mathbf{1.2846 \cdot 10^{-6}}$

Table 6.4: Results of the different optimization techniques for the 10D optimal control problem.

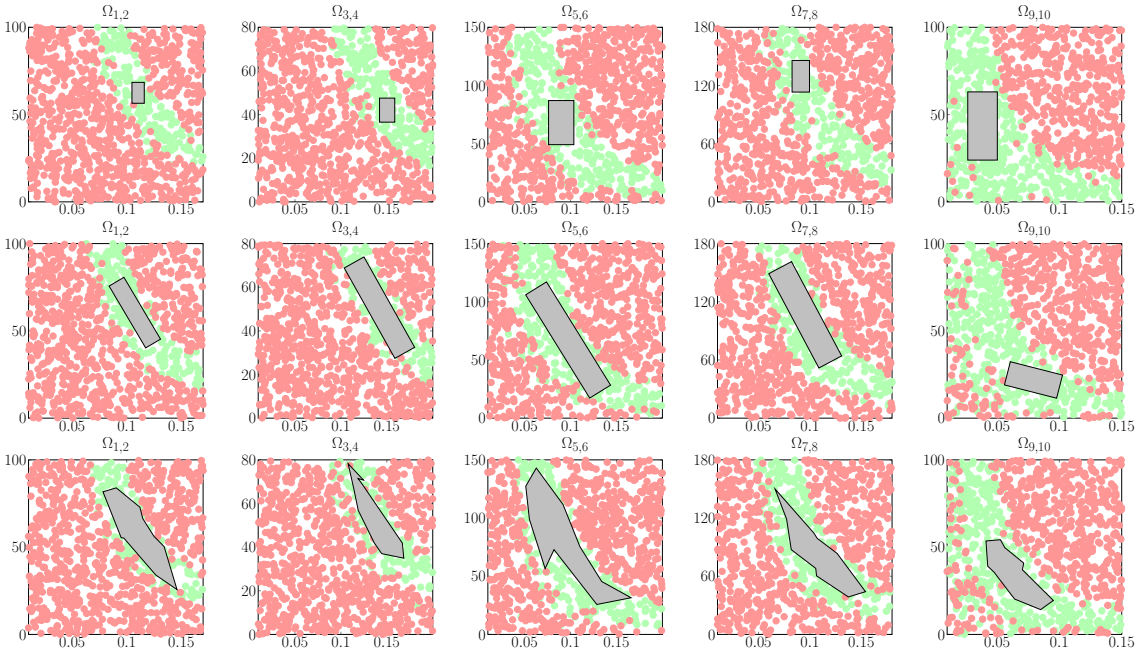

Figure 6.8: Solution spaces with volume close to the mean for the 10D optimal control problem.

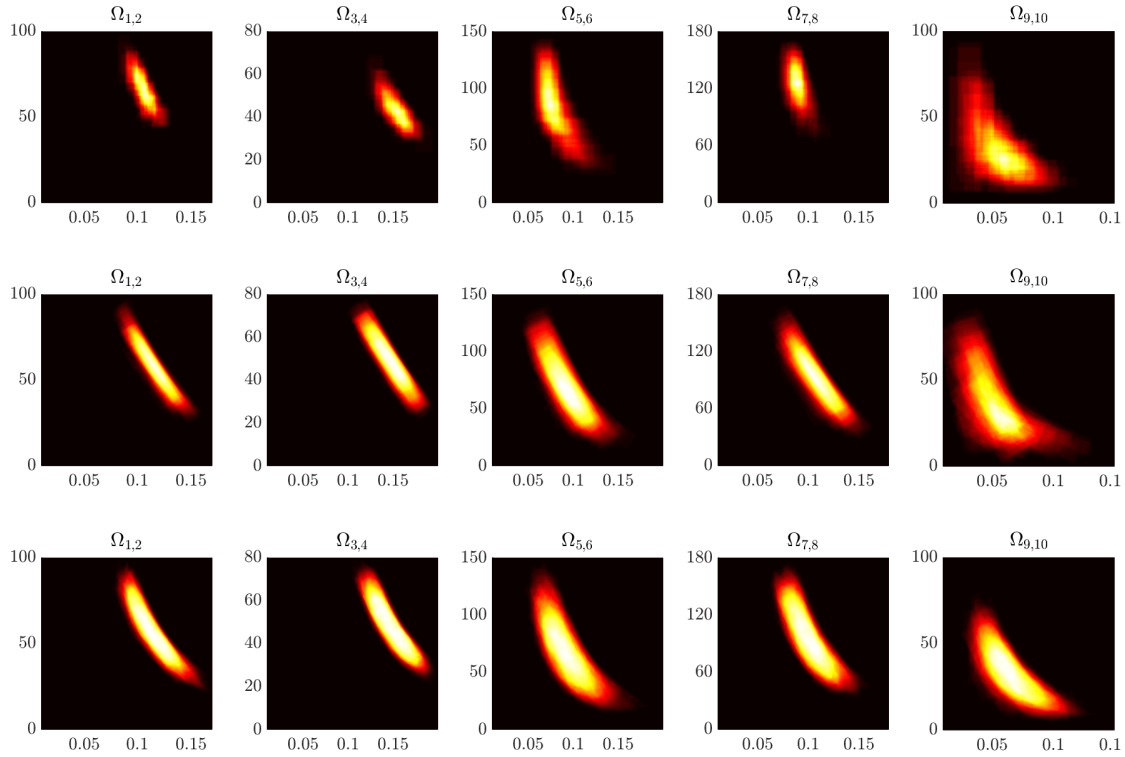

Figure 6.9: Heat maps for the 10D optimal control problem.

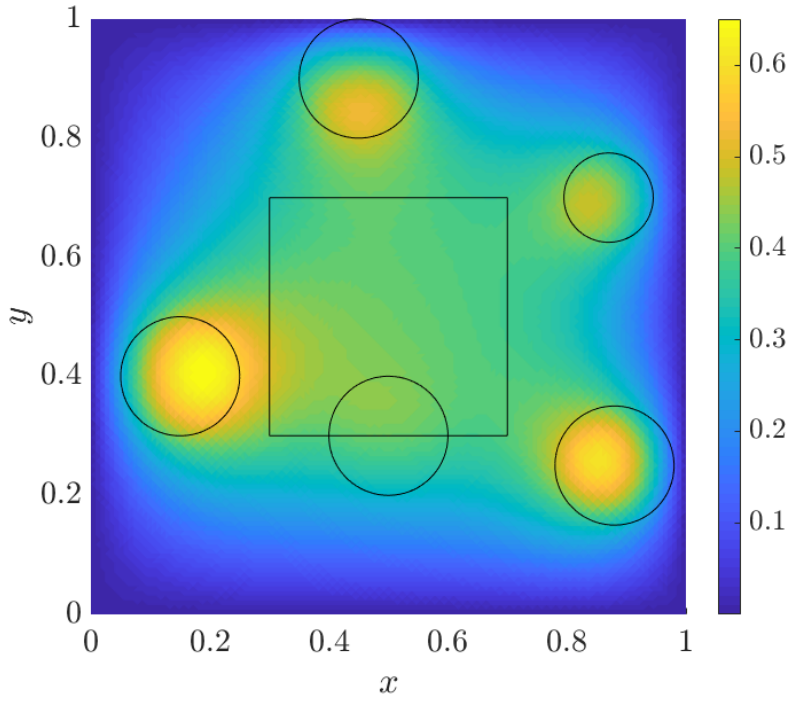

Figure 6.10: A solution of the heat equation represented by a design point taken from within a polytope. The region K is marked with a black square and the radii of the heat sources are marked with black circles.

6.4 Study of Sample Sizes

Finally, we are interested in the impact of the number of sample points on the results of the box optimization algorithm, the rotated box optimization algorithm and the polytope optimization algorithm. We have briefly mentioned this topic in Subsection 2.3.2, where we found the estimate $N = 100$ for the number of sample points such that the box optimization algorithm yields solution spaces with a high enough quality. However, so far we have not yet verified this number through numerical experiments. Additionally, we have tacitly assumed that this number is applicable to the rotated box optimization algorithm and the polytope optimization algorithm. Thus, in this section we apply multiple sample sizes to the box optimization algorithm, the rotated box optimization algorithm and the polytope optimization algorithm.

In practice, we are generally not able to sample an arbitrary number of design points, as an evaluation of the objective function f can be very expensive. Thus, we limit the total amount of designs that can be sampled by one instance of each optimization algorithm to a budget of $B := 20000$. We distribute this budget evenly between the exploration and the consolidation phase, such that we sample at most 10000 designs in each phase. If we increase the number of designs that are sampled in each step of a phase, we have to decrease the total number of steps done in that phase. Thus, we test the optimization algorithms with the following four configurations:

1) $N = 100$, $n_{\text{exp}} = n_{\text{con}} = 100$, which is the standard configuration from the previous experiments,

2) $N = 200$, $n_{\text{exp}} = n_{\text{con}} = 50$,

3) $N = 500$, $n_{\text{exp}} = n_{\text{con}} = 20$, and

4) $N = 1000$, $n_{\text{exp}} = n_{\text{con}} = 10$.

We apply these configurations to the box optimization algorithm, the rotated box optimization algorithm and the polytope optimization algorithm, which we then test for the 4D Rosenbrock function, the 6D linear problem from vehicle dynamics (see Section 5.2), the 8D acoustics problem and the 10D problem from optimal control. The results can be found in Tables 6.5, 6.6, 6.7 and 6.8. The best result from each algorithm is printed in bold.

Apparently, the box optimization algorithm does not profit at all from fewer steps in the exploration and consolidation phases, despite an increased number of sample points. For example, the mean normalized volume found with the fourth configuration applied to the 4D Rosenbrock function is only 4% of the volume found with the first configuration. In all other cases, the non-standard configurations also find less volume than the standard configuration.

The rotated box optimization algorithm leads to mixed results. The configurations result in more or less the same volume for the 4D Rosenbrock function, more volume for the 6D vehicle dynamics problem and less volume for the 10D optimal control problem. For the 8D acoustics problem, the second and third configuration are better than the standard configuration, however, the fourth configuration is worse.

Finally, for the polytope optimization algorithm, the second and third configuration are always better than the first. The third configuration even yields double and triple the volume for the 8D acoustics problem and the 10D optimal control problem. The fourth configuration has a mixed performance, being much worse than the standard configuration for the 4D Rosenbrock function, better than the standard configuration, but worse than the second or third configuration for the 8D acoustics problem, and better than all other configurations for the 6D vehicle dynamics problem.

In conclusion, increasing the number of sampled designs and decreasing the number of steps seems to be unfavorable for the box optimization algorithm. This is, to some extent, in line with the results from Subsection 2.3.2, where it has been stated that 100 sampled designs are enough to ensure a high quality of the solution space. The consequences are unclear for the rotated box optimization algorithm, as adjusting the number of sampled designs and steps may or may not help to increase the volume of the solution spaces. For the polytope optimization algorithm, configurations 2 and 3 seem to be better overall than the standard configuration. Consequently, one of these configurations should be used when the polytope optimization algorithm is applied to a problem. Configuration 4 is mostly worse than the others, likely because only 10 steps are not enough to find a large amount of good design space in the exploration phase. Hence, it should not be used.

4D Rosenbrock

Algorithm	$N = 100$ 100 steps	$N = 200$ 50 steps	$N = 500$ 20 steps	$N = 1000$ 10 steps
Box Optimization	**0.0077**	0.0059	0.0012	0.0003
Rotated Box Optimization	**0.0078**	**0.0078**	0.0072	0.0072
Polytope Optimization	0.0399	**0.0437**	0.0434	0.0155

Table 6.5: Mean normalized volumes of the algorithms for the 4D Rosenbrock problem and multiple sample sizes.

6D Vehicle Dynamics

Algorithm	$N = 100$ 100 steps	$N = 200$ 50 steps	$N = 500$ 20 steps	$N = 1000$ 10 steps
Box Optimization	$\mathbf{0.3522 \cdot 10^{-4}}$	$0.3211 \cdot 10^{-4}$	$0.3103 \cdot 10^{-4}$	$0.2996 \cdot 10^{-4}$
Rotated Box Optimization	$1.6167 \cdot 10^{-4}$	$1.7998 \cdot 10^{-4}$	$1.8258 \cdot 10^{-4}$	$\mathbf{1.9415 \cdot 10^{-4}}$
Polytope Optimization	$2.0232 \cdot 10^{-4}$	$2.5875 \cdot 10^{-4}$	$3.3436 \cdot 10^{-4}$	$\mathbf{3.4950 \cdot 10^{-4}}$

Table 6.6: Mean normalized volumes of the algorithms for the 6D problem from vehicle dynamics and multiple sample sizes.

8D Acoustics

Algorithm	$N = 100$ 100 steps	$N = 200$ 50 steps	$N = 500$ 20 steps	$N = 1000$ 10 steps
Box Optimization	$\mathbf{0.0060 \cdot 10^{-5}}$	$0.0036 \cdot 10^{-5}$	$0.0039 \cdot 10^{-5}$	$0.0041 \cdot 10^{-5}$
Rotated Box Optimization	$0.1966 \cdot 10^{-5}$	$0.2885 \cdot 10^{-5}$	$\mathbf{0.3432 \cdot 10^{-5}}$	$0.1353 \cdot 10^{-5}$
Polytope Optimization	$0.5575 \cdot 10^{-5}$	$0.8372 \cdot 10^{-5}$	$\mathbf{1.1673 \cdot 10^{-5}}$	$0.6690 \cdot 10^{-5}$

Table 6.7: Mean normalized volumes of the algorithms for the 8D problem from acoustics and multiple sample sizes.

10D Optimal Control

Algorithm	$N = 100$ 100 steps	$N = 200$ 50 steps	$N = 500$ 20 steps	$N = 1000$ 10 steps
Box Optimization	$\mathbf{0.0045 \cdot 10^{-6}}$	$0.0030 \cdot 10^{-6}$	$0.0029 \cdot 10^{-6}$	$0.0029 \cdot 10^{-6}$
Rotated Box Optimization	$\mathbf{1.1636 \cdot 10^{-6}}$	$1.0849 \cdot 10^{-6}$	$1.0888 \cdot 10^{-6}$	$0.2773 \cdot 10^{-6}$
Polytope Optimization	$3.2493 \cdot 10^{-6}$	$7.5522 \cdot 10^{-6}$	$\mathbf{9.6046 \cdot 10^{-6}}$	$5.3048 \cdot 10^{-6}$

Table 6.8: Mean normalized volumes of the algorithms for the 10D optimal control problem and multiple sample sizes.

Chapter 7

Conclusion

In this thesis, we developed the rotated box optimization algorithm and the polytope optimization algorithm, which are variations of the box optimization algorithm. All three algorithms can be employed in a set-based design process.

The conclusions from this thesis can be summarized as follows:

- The coupling of parameters on 2D-maps, which is the underlying principle of the rotated box optimization algorithm and the polytope optimization, proved to be a helpful tool for handling problems from set-based design. Hence, techniques that are enhanced with 2D-maps may yield good results in any context where the coupling of a few parameters is tolerable.
- With the help of 2D-maps, we were able to generalize the box optimization algorithm. We replaced axis-parallel boxes on 2D-maps by rotated boxes, which led to the rotated box optimization algorithm. Then, we entirely abandoned the premise of box-shaped solution spaces and introduced polygon-shaped solution spaces. With these, we were able to construct the polytope optimization algorithm.
- All three algorithms work in high dimensions despite the curse of dimensionality, as demonstrated in Chapters 5 and 6. This makes them applicable to a wide range of problems.
- There are many parameters that determine the outcome of the algorithms. We were able to identify a few parameter settings that seem to be universal.
- Additionally, with the given parameter settings, the algorithms are robust. They will not quit unexpectedly and will always give a solution space as output.
- We observed in Chapters 5 and 6 that the rotated box optimization algorithm and the polytope optimization algorithm generally yield better results than the box optimization algorithm. This means that, if possible, one of those two algorithms should be applied instead of the box optimization algorithm.
- Most importantly, the rotated box optimization algorithm and the polytope optimization algorithm do not require more design samples to achieve better

results. They are much more cost-efficient than the box optimization algorithm in the sense that their ratio of the number of design samples to the volume of the final box is larger.

- Finally, as shown in Section 6.4, adjusting the number of design samples per step may further improve the results of the rotated box optimization algorithm and the polytope optimization, whereas the box optimization algorithm does not profit at all from such an adjustment.

As a consequence, if one wishes to apply solution spaces for a robust design process and a coupling of design parameters is feasible, one should at least use the rotated box optimization algorithm because it is not much more difficult to implement than the box optimization algorithm. However, the polytope optimization algorithm yields better results and is more consistent. It should hence be preferred over the other two algorithms.

Possible future work might involve the coupling of three design variables instead of two. The design space could then be expressed as a product of 3D-maps. Consequently, we would need to find a solution space that is represented by a product of 3D rotated boxes or 3D polytopes. Of course, this idea could be extended to couple an arbitrary number of n design variables. A solution space would then be the product of n-dimensional rotated boxes or n-dimensional polytopes. For some applications, it might even be desirable to couple different numbers of design variables, i.e., two parameters have to be coupled on a 2D-map, while five other parameters have to be coupled on a 5D-map.

Finally, it might prove useful to investigate differently shaped solution spaces. They could, for example, be represented by a closed curve or by a classifier obtained from a support vector machine. Their shape might be more flexible than that of a polygon, which could allow them to find more good design space. Solution spaces that consist of multiple connected components might also find much more good design space, simply because they could, theoretically, split up and explore different regions of the design space. These new types of solution spaces will require new ideas for trimming, growing and evaluation.

Bibliography

[1] C. Audet and W. Hare. *Derivative-Free and Blackbox Optimization*. Springer International Publishing AG, Cham, 2017.

[2] M. Beer and M. Liebscher. Designing robust structures — A nonlinear simulation based approach. *Computers & Structures*, 86(10):1102–1122, 2008.

[3] A. Ben-Tal and A. Nemirovski. *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*. MOS-SIAM Series on Optimization, Society for Industrial and Applied Mathematics and Mathematical Programming Society, Philadelphia, 2001.

[4] J.M. Bernardo and A.F.M. Smith. *Bayesian Theory*. John Wiley & Sons, Chichester, 2000.

[5] H.-G. Beyer and B. Sendhoff. Robust optimization — A comprehensive survey. *Computer Methods in Applied Mechanics and Engineering*, 196(33–34):3190–3218, 2007.

[6] J.C. Bezdek, R. Ehrlich and W. Full. FCM: The fuzzy c-means clustering algorithm. *Computers & Geosciences*, 10(2–3):191–203, 1984.

[7] C.M. Bishop. *Pattern Recognition And Machine Learning*. Springer, New York, 2009.

[8] J.K. Blitzstein and J. Hwang. *Introduction to Probability*. CRC Press, Taylor & Francis Group, Abingdon, 2014.

[9] A.J. Booker, J.E. Dennis, P.D. Frank, D.B. Serafini, V. Torczon and M.W. Trosset. A rigorous framework for optimization of expensive functions by surrogates. *Structural Optimization*, 17(1):1–13, 1999.

[10] D.S. Broomhead and D. Lowe. Multivariable Functional Interpolation and Adaptive Networks. *Complex Systems*, 2(3):321–355, 1988.

[11] C.G. Broyden. The Convergence of a Class of Double-Rank Minimization Algorithms. *IMA Journal of Applied Mathematics*, 6(1):76–90, London, 1970.

[12] F. Campolongo, M. Ratto, A. Saltelli and S. Tarantola. *Sensitivity Analysis in Practice: A Guide to Assessing Scientific Models*. John Wiley & Sons, Chichester, 2004.

[13] A.L. Cauchy. Méthode générale pour la résolution des systèmes d'équations simultanées. *Œuvres complètes*, série 1, tome 10, 399–402. Comptes rendus de l'Académie des sciences, 25:536–538. Paris, 1847.

[14] A. Chatterjee. An introduction to the proper orthogonal decomposition. *Current Science*, 78(8):808–817, 2000.

[15] W. Chen and C. Yuan. A probabilistic-based design model for achieving flexibility in design. *Journal of Mechanical Design*, 121(1):77–83, 1999.

[16] K.K. Choi, B.D. Youn and R.-J. Yang. Moving least square method for reliability-based design optimization. *Fourth World Congress of Structural and Multidisciplinary Optimization*, Dalian, China, 2001.

[17] J.H. Conway, H. Burgiel and C. Goodman-Strauss. *The Symmetries of Things*. A. K. Peters, Wellesley, 2008.

[18] P.K. Das, D. Faulkner and Y. Pu. A strategy for reliability-based optimization. *Engineering Structures*, 19(3):276–282, 1997.

[19] W.C. Davidon. Variable metric method for minimization. *SIAM Journal on Optimization*, 1(1):1–17, 1991.

[20] A. Der Kiureghian and O. Ditlevsen. Aleatory or epistemic? Does it matter? *Structural Safety*, 31(2):105–112, 2009.

[21] F. Duddeck and E. Wehrle. Recent advances on surrogate modeling for robustness assessment of structures with respect to crashworthiness requirement. *10th European LS-DYNA Conference*, Würzburg, Germany, 2015.

[22] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, 1942–1948, IEEE, 1995.

[23] M. Eichstetter, S. Müller and M. Zimmermann. Product family design with solution spaces. *Journal of Mechanical Design*, 137(12):121401, 2015.

[24] S. Erschen. *Optimal Decomposition of High-Dimensional Solution Spaces for Chassis Design*. PhD Thesis, Department of Civil, Geo and Environmental Engineering, Technical University of Munich, Germany, 2017.

[25] S. Erschen, F. Duddeck, M. Gerdts and M. Zimmermann. On the optimal decomposition of high-dimensional solution spaces of complex systems. *ASME Journal of Risk and Uncertainty in Engineering Systems, Part B: Mechanical Engineering*, 4(2):021008, 2017.

[26] J. Fender. *Solution Spaces for Vehicle Crash Design*. PhD Thesis, Department of Civil, Geo and Environmental Engineering, Technical University of Munich, Germany, 2013.

[27] J. Fender, F. Duddeck and M. Zimmermann. Direct computation of solution spaces. *Structural and Multidisciplinary Optimization*, 55(5):1787–1796, 2017.

[28] W.W. Finch and A.C. Ward. Quantified Relations: A class of predicate logic design constraints among sets of manufacturing, operating, and other variations. *Proceedings of The 1996 ASME Design Engineering Technical Conferences and Computers in Engineering Conference*, Irvine, 1996.

[29] R. Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317–322, 1970.

[30] R. Fletcher and M.J.D. Powell. A rapidly convergent descent method for minimization. *The Computer Journal*, 6(2):163–168, 1963.

[31] G. Fung, S. Sandilya and R.B. Rhao. Rule extraction from linear support vector machines. *Rule Extraction from Support Vector Machines. Studies in Computational Intelligence*, 80:83–107. Springer, Berlin–Heidelberg, 2008.

[32] A. Gelman, J.B. Carlin, H.S. Stern and D.B. Rubin. *Bayesian Data Analysis.* Chapman & Hall/CRC, Boca Raton, 2004.

[33] M. Gendreau and J.-Y. Potvin. *Handbook of Metaheuristics.* Springer, New York, 2010.

[34] H.-O. Georgii. *Stochastics. Introduction to Probability and Statistics.* De Gruyter, Berlin–Boston, 2012.

[35] R.G. Ghanem and P.D. Spanos. *Stochastic Finite Elements. A Spectral Approach.* Dover Publications, New York, 2003.

[36] M. Götz, M. Liebscher and W. Graf. Efficient detection of permissible design spaces in an early design stage. *11. LS–DYNA Forum*, 184–185, 2012.

[37] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading–Massachusetts–Boston, 1989.

[38] D. Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of Computation*, 24(109):23–26, 1970.

[39] W. Graf, M. Götz and M. Kaliske. Computing permissible design spaces under consideration of functional responses. *Advances in Engineering Software*, 117:95–106, 2018.

[40] L. Graff. *A Stochastic Algorithm for the Identification of Solution Spaces in High-Dimensional Design Spaces.* PhD thesis, Faculty of Science, University of Basel, Switzerland, 2013.

[41] L. Graff, J. Fender, H. Harbrecht and M. Zimmermann. Identifying key parameters for design improvement in high-dimensional systems with uncertainty. *Journal of Mechanical Design*, 136(4):041007, 2014.

[42] L. Graff, H. Harbrecht and M. Zimmermann. On the computation of solution spaces in high dimensions. *Structural and Multidisciplinary Optimization*, 54(4):811–829, 2016.

[43] R.L. Graham. An efficient algorith for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132–133, 1972.

[44] H. Harbrecht, D. Tröndle and M. Zimmermann. A sampling-based optimization algorithm for solution spaces with pair-wise coupled design variables. *Structural and Multidisciplinary Optimization*, 60:501–512, 2019.

[45] H. Harbrecht, D. Tröndle and M. Zimmermann. Approximating solution spaces as a product of polygons. Preprint 2019–13, Fachbereich Mathematik, Universität Basel, 2019.

[46] J.H. Holland. *Adaptation in Natural and Artificial Systems*. The MIT Press, Cambridge, 1993.

[47] K. Hormann and A. Agathos. The point in polygon problem for arbitrary polygons. *Computational Geometry*, 20:131–144, 2001.

[48] A.K. Jain. Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, 31(8):651–666, 2010.

[49] F. Jarre and J. Stoer. *Optimierung*. Springer, Berlin–Heidelberg, 2019.

[50] I.T. Jolliffe. *Principal Component Analysis*. Springer, New York, 1986.

[51] V. Kitov. Dimensionality reduction. *Summer School on Machine Learning in High Energy Physics*, Deutsches Elektronen-Synchrotron DESY, Hamburg, 2015.

[52] D.G. Krige. *A statistical approach to some mine valuation and allied problems on the Witwatersrand*. Master's Thesis, University of the Witwaterstrand, South Africa, 1951.

[53] M. Lehar and M. Zimmermann. An inexpensive estimate of failure probability for high-dimensional systems with uncertainty. *Structural Safety*, 36–37:32–38, 2012.

[54] R.J. Malak Jr., J.M. Aughenbaugh and C.J.J. Paredis. Multi-attribute utility analysis in set-based conceptual design. *Computer-Aided Design*, 41:214–227, 2009.

[55] G. Matheron. Principles of geostatistics. *Economic Geology*, 58(8):1246–1266, 1963.

[56] M. Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, Cambridge, 1996.

[57] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1000, 1997.

[58] Y.-E. Nahm and H. Ishikawa. Novel space-based design methodology for preliminary engineering design. *The International Journal of Advanced Manufacturing Technology*, 28(11–12):1056–1070, 2006.

[59] J.A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.

[60] M.S. Nixon and A.S. Aguado. *Feature Extraction and Image Processing*. Elsevier/Academic Press, Amsterdam–Boston, 2008.

[61] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer, New York, 2006.

[62] W. Pedrycz and F. Gomide. *An Introduction to Fuzzy Sets*. The MIT Press, Cambridge, 1998.

[63] A. Piotrow, M. Liebscher, S. Pannier and W. Graf. Grouping detection of uncertain structural process by means of cluster analysis. *7th European LS–DYNA Conference*, 2014.

[64] N.V. Queipo, R.T. Haftka, W. Shyy, T. Goel, R. Vaidyanathan and P.K. Tucker. Surrogate-based analysis and optimization. *Progress in Aerospace Sciences*, 41(1):1–28, 2005.

[65] C.M. Rocco, J.M. Moreno and N. Carrasquero. Robust design using a hybrid-cellular-evolutionary and interval-arithmetic approach: a reliability application. *Reliability Engineering and System Safety*, 79:149–159, 2003.

[66] J. Sacks, S.B. Schiller and W.J. Welch. Designs for computer experiments. *Technometrics*, 31(1):41–47, 1989.

[67] R. Schaback and H. Wendland. *Numerische Mathematik*. Springer, Berlin–Heidelberg, 2005.

[68] D.F. Shanno. Conditioning of quasi-Newton methods for function minimization. *Mathematics of Computation*, 24(111):647–656, 1970.

[69] J.K. Sharma. *Business Statistics*. Pearson India, New Delhi, 2006.

[70] Y. Shi and R. Eberhart. A modified particle swarm optimizer. *1998 IEEE International Conference on Evolutionary Computation Proceedings*, 69–73, 1998.

[71] D.J. Singer, N. Doerry and M.E. Buckley. What is set-based design? *Naval Engineers Journal*, 121(4):31–43, 2009.

[72] D. Sobek, A.C. Ward and J. Liker. Toyota's principles of set-based concurrent engineering. *Sloan Management Review*, 40(2):67–83, 1999.

[73] R. Storn and K. Price. Differential evolution — a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[74] D. Sunday. *Inclusion of a Point in a Polygon*. http://geomalgorithms.com/a03-_inclusion.html. December 16, 2019.

[75] J.A.K. Suykens. Nonlinear modelling and support vector machines. *Conference Record — IEEE Instrumentation and Measurement Technology Conference*, 1(1):287–294, 2001.

[76] R. Szeliski. *Computer Vision. Algorithms And Applications*. Springer, London–New York, 2011.

[77] F. Tröltzsch. *Optimale Steuerung Partieller Differentialgleichungen*. Vieweg + Teubner, Wiesbaden, 2009.

[78] J. Tu, K.K. Choi and Y.H. Park. A new study on reliability-based design optimization. *Journal of Mechanical Design*, 121(4):557–564, 1999.

[79] Laboratory Test Procedure for New Car Assessment Program Frontal Impact Testing. U.S. Department of Transportation, National Highway Traffic Safety Administration, Office of Vehicle Safety, Office of Crashworthiness Standards, 2012.

[80] A.R. Webb and K.D. Copsey. *Statistical Pattern Recognition*. Wiley, Hoboken, 2011.

[81] S.S.A. Willaert, R. de Graaf and S. Minderhoud. Collaborative engineering: a case study of concurrent engineering in a wider context. *Journal of Engineering and Technology Management*, 15:87–109, 1998.

[82] R.I. Winner, J.P. Pennell, H.E. Bertrand and M.M.G. Slusarczuk. The role of concurrent engineering in weapons system acquisition. *Institute for Defense Analyses, Report R-338*, 1988.

[83] S.J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

[84] B.D. Youn, K.K. Choi, R.-J. Yang and L. Gu. Reliability-based design optimization for crashworthiness of vehicle side impact. *Structural and Multidisciplinary Optimization*, 26:272–283, 2004.

[85] M. Zimmermann and J.E. von Hoessle. Computing solution spaces for robust design. *International Journal for Numerical Methods in Engineering*, 94(3):290–307, 2013.

[86] M. Zimmermann, S. Königs, C. Niemeyer, J. Fender, C. Zeherbauer, R. Vitale and M. Wahle. On the design of large systems subject to uncertainty. *Journal of Engineering Design*, 28(4):233–254, 2017.

[87] P. Zörnig. *Nonlinear Programming*. De Gruyter, Berlin–Boston, 2014.

Curriculum Vitae

Personal Data

Name	Dennis Thassilo Tröndle
Date of Birth	23rd of June 1990
Place of Birth	Bad Säckingen, Germany
Nationality	German

Education

1996 – 2000	Primary School
	Grundschule Nöggenschwiel
2000 – 2009	High School
	Klettgau-Gymnasium Tiengen
2009 – 2013	BSc Student in Mathematics
	University of Basel
2013 – 2015	MSc Student in Mathematics
	University of Basel
2015 – 2020	PhD Student in Mathematics
	University of Basel