Soft Computing Methods for Applied Shape Optimization

Vom Fachbereich Maschinenbau
der Technischen Universität Darmstadt
zur Erlangung des Grades eines Doktor-Ingenieurs
(Dr.-Ing.)
genehmigte Dissertation

von

Diplom-Mathematiker Kai Hirschen, M. Sc.
aus Zell/Mosel

Berichterstatter:	Prof. Dr. rer. nat. M. Schäfer
Mitberichterstatter:	Prof. Dr. rer. nat. A. Martin
Tag der Einreichung:	16.08.2004
Tag der mündlichen Prüfung:	19.10.2004

D 17

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe. Alle Stellen, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen worden sind, sind durch Angaben der Quelle deutlich gekennzeichnet. Ich erkläre außerdem, dass ich noch keinen Promotionsversuch unternommen habe.

Darmstadt, den 04. November 2004

Kai Hirschen

“Viel Denken, nicht viel Wissen soll man pflegen”

Demokrit

Preface

I feel indebted and like to express my gratitude to Professor Michael Schäfer for supervising me during my time at the Department of Numerical Methods in Mechanical Engineering. I especially acknowledge for granting me scientific freedom to do research in this topic of my special interest. Sincere thanks go to Professor Alexander Martin who kindly accepted to be co-examiner of this work.

My thanks also go to Dipl.-Ing. Florian Schmid and Dr.-Ing. Sebastian Meynen who provided the structural mechanics computations in the joint paper [135]. Beyond the fruitful cooperation, my special thanks go to Sebastian for helping out many times in any kind of issue related to this dissertation.

I also appreciate the Deutsche Forschungsgemeinschaft (DFG) for financial support in the Graduiertenkolleg 853 “Modellierung, Simulation und Optimierung von Ingenieuranwendungen”.

Contents

Chapter 1

Introduction

Soft computing is the computer science part of bionics where the understanding and transformation of principles from nature into engineering problems are investigated. The role model for soft computing is the functioning of the human mind which is characterized in aspects that it is tolerant of imprecision, uncertainty and partial truth, hence, the name 'soft computing'. The guiding principle of soft computing is to exploit these properties to achieve tractability, robustness and low solution cost. Principal constituents are evolutionary algorithms, neural computing, Fuzzy logic, machine learning and probabilistic reasoning in all of which the functioning of biological processes become evident. Neural networks, for instance, model the way humans process information through the natural neural system. Soft computing methods can be used to render optimization problems which are highly complex and where conventional optimization procedures quickly fail to provide satisfactory results within reasonable time. This dissertation emphasizes on the realization of design improvements for engineering applications which are hardly tractable by conventional optimization procedures.

Optimization procedures are fairly impossible to be accomplished without the help of computers. The efforts and costs in aspect of time and money of constructing and building diverse design configurations is unbearable. Hence, optimization presumes the capability to describe physical phenomena in a computational environment. On the other hand, the optimization procedure by itself has again to be executed in a computational vicinity. It is commonly acknowledged, that by the aid of computers, development time for products may be accelerated and at the same time, a higher quality of product features can be achieved. There is a clear trend in industry towards more complex products spanning over several scientific disciplines. This process is made possible by the fast advancement of computer processing power and memory storage potentials throughout the last decade. This trend is only likely to grow more pronounced in the future as computers become increasingly cheaper and more powerful while traditional forms of testing become increasingly expensive.

However, we are currently at the very first stages to replace real world experiments with computer simulations. Research about strategies to optimize real-world problems in a

computer simulation environment just began one decade ago. Earlier, mathematicians proposed many efficient algorithms to render optimization problems by finding an extremum for a given function. The main drawback with these mathematical approaches is that they are mainly designed and primarily applied to 'theoretic' problems, expressed as formulae in closed form. Now reality, in this context, is not described by closed formulas but rather by complex models which only allow approximations to their solution.

This work is a highly multi-disciplinary project, it involves concepts and methods from mathematics, computer science and engineering. The strength of this work is to merge ideas, concepts and methods from these scientific areas and illustrate an optimization framework that is particularly tailored for practical engineering problems. As an exemplary test problem we consider the optimization of a heat exchanger configuration with respect to several objectives like pressure drop, maximum temperature, and covered flow area. These are conflicting objectives and thus the optimization procedure can quickly become infeasible for conventional routines like gradient descent algorithms. Another example is the minimization of pressure drop for a staggered pipe system. Each evaluation run is computationally expensive and there is no a priori information available concerning the functional coherence between design parameters and objective function. It is thus appropriate to apply an approximation model which is able to depict this functional coherence and at the same time serves as a computational cheap surrogate to the numerical solver. Once the approximation model is set up and design evaluations become computationally cheap, an arbitrary number of function calls are available for any optimization routine.

1.1 Motivation

It is a very demanding and only recently mastered task to correctly depict physical behavior on a computer. In general, physical phenomena can be described by an abstract mathematical model expressed in differential equations. The solutions of these differential equations cannot be formulated in closed form, so one has to resort to numerical methods to produce approximations to the actual solutions.

After the solution becomes educible, it is desired to choose design parameters in such a way that the physical system behaves optimal according to certain criteria. This optimization process is facilitated by an optimization model and an appropriate optimization algorithm. Whereas optimization is a classical mathematical discipline, the desire to improve an objective is to be found in many scientific fields. We here deal with parametrized flow geometries as they appear in mechanical engineering context. However, the application of mathematical algorithms is not straightforward since these are most often not designed for practical needs. Thus, special considerations are necessary to understand which algorithms are usable for specific applications. We will discuss salient issues we are faced with when optimizing flow geometries.

Optimization is difficult since mathematical routines usually work with some kind of gradient information. One of the salient attributes in fluid mechanics is that there is

no derivative easily available. Though this derivative can be approximated, it can be very computationally expensive to obtain gradient information and it has to be paid by time-consuming function evaluations. This gives rise to the usage of optimization routines which do not require gradient information. One approach is to build substitute models, as response surface methods do. However, the optimization needs to be highly efficient to achieve a result in feasible time.

Conventional optimization methods like gradient descent only yield satisfiable results under certain assumptions on properties of the objective function, e.g. convexity, smoothness, unimodality etc. Real-world engineering tasks faces new challenges. The optimization problem then usually exhibits non-convex behavior, multiple local minima, nonlinearities, a large parameter and output space dimension, and there are several conflicting objectives to be considered. For such complex optimization problems, even mathematically well known gradient methods in fact are considered to be not the best choice.

1.2 State of the Art

There is a huge number of optimization algorithms available where most of them solve a particular optimization problem with pre-defined attributes. We here put a special focus on evolutionary algorithms which are then used in conjunction with approximation models and also play a crucial role in multi-objective algorithms.

Industry practice [103,124,125,148] as well as academic research [16,17,76,113] has shown that, in principle, an optimization approach based on stochastic principles may be superior to conventional optimization strategies. Yet, corresponding stochastic simulation algorithms are commonly not applicable since the computation is too much time-consuming. This can be circumvented since recent research efforts in this field proposed to use specific evolutionary operators. Since this methodology is at its cutting edge of research, the mentioned methods are not yet popular throughout the engineering community. Nevertheless, the usage of optimization in engineering design is gaining wider acceptance in all fields of industry and research as the computational capabilities increase. The following is a survey and a reference of contemporary literature in the respective field of evolutionary algorithms, multi-objective optimization, approximation models, and numerical simulation.

Evolutionary Algorithms Evolutionary Algorithms (EA) are general-purpose search procedures based on the mechanisms of natural selection and population genetics. They spread parameters in the design space and thus ensure a global search based on stochastic operators. Usually grouped under the term evolutionary algorithms, we divide the domains of genetic algorithms, evolutionary strategies, evolutionary and genetic programming.

John Holland [67] was the pioneering founder of genetic algorithms. His algorithms work on bitstrings encoding a parametrized system. In 1972, Rechenberg [123] introduced evo-

lutionary strategies and used them for improving an airfoil design. The salient difference between genetic algorithms and evolutionary strategies is the representation of parameters. Evolutionary strategies work on a continuously parametrized problem, as is typical in shape design optimization problems, and are thus more appropriate for this purpose. For completeness, we mention genetic programming [88] which is a related technique in where computer programs, rather than function parameters, are optimized. Genetic programming often uses tree-based internal data structures to represent the computer programs for adaption.

According to a the particular task an EA has to solve, several operators and strategies can be considered. In case of multi-modal problems, sharing operators penalty individuals which crowd in local minima, hence ensuring a population spread which maximizes information in the group. For multi-objective problems, variants of these sharing operators are used to provide a sufficient spread along the Pareto front. However, evolutionary optimization techniques have not yet found widespread acceptance for industrial purposes. This stems from the fact that, in the single-objective case, evolutionary algorithms involve more (expensive) function evaluations of the objective and, they are less accurate than gradient based methods. In multi-objective problems, the mentioned drawbacks can become negligible in comparison to the benefits gained. Evolutionary Algorithms can also be used for demanding problems like the traveling salesman problem and the knapsack problem (cf. [139]), both of which belong to the computational complexity class of NP-hard problems and are not tractable by conventional gradient descent methods. Today, branch and cut algorithms (cf. [4,5]) and variants thereof are used to solve the traveling salesman problem with no less than 13 000 cities. Up to today, EA found applications in many areas, including computer science, architectural and civil engineering, electrical, control and signal processing, mechanical and industrial engineering, as well as in economics, operations research, ecology, population genetics, social systems and many other fields. Fogel [46] and Michalewicz [104] discuss many real-life problems in which heuristic techniques like EA solve demanding problems. Goldberg [53], Bäck [7], Deb [26, 30, 31] and Eiben et al. [39, 40] introduce in EA methodology and highlight EA applications. Bäck [7] is also an extensive reference and covers EA convergence issues. For engineering purposes, Schütz and Schwefel [138] and Jakiela et al. [80] demonstrated the usefulness of EA. In [121], a large compilation of papers is presented where EA are used for engineering relevant problems. Duvigneau and Visonneau [37] and Giannakoglou [52] and Giannakoglou et al. [85] are recent tracts on EA in conjunction with artificial neural networks to render design optimization problems in a CFD environment. Deb et al. [28,32,34] is a contemporary account for EA solving parameterized real-world applications. The ERCOFTAC [42] as well as the ECCOMAS [44] conference proceedings offer recent contributions in the application of evolutionary algorithms for engineering related optimization problems.

Multi-Objective Optimization In multi-objective optimization, the aim is to optimize according to multiple criteria, all of similar importance. Usually, this is achieved by applying evolutionary algorithm operators and manipulating them such that they produce a number of equivalent (or: equally optimal) solutions.

Several methods are proposed to approximate the Pareto-front, a set of solutions representing non-dominated designs. The actually first algorithm to accomplish optimizing in multiple conflicting objectives was proposed in 1984 by Schaffer [132] who applied a sorting of solutions according to their corresponding objectives. In 1989, Goldberg [53] established a multi-objective optimization algorithm based on the Pareto dominance concept. His algorithm was the initialization for several methods proposed in the 1990's.

Due to their diversity preserving features, variation operators can eventually destroy the best individuals in the population. To avoid this, elitism is incorporated in the selection operators. By carrying the best solutions into the next generation, it ensures that these individuals are always retained in the population. Furthermore, not all but many algorithms which employ elitism can be shown to be convergent. Well known in this aspect are

- Knowles and Corne's Pareto archived evolution strategy (PAES) [87]
- Hajela and Lin's weighting-based genetic algorithm [60]
- Fonseca and Fleming's multiobjective genetic algorithm (MOGA) [47]
- Horn, Nafpliotis, and Goldberg's niched Pareto genetic algorithm (NPGA) [68, 69]
- Srinivas and Deb's nondominated sorting genetic algorithm (NSGA) [144].

Overviews of evolutionary multi-objective algorithms are given in Fonseca and Fleming [48], Coello [20], Zitzler et al. [162] and Deb [25]. Theoretical aspects like convergence issues can be found, e.g., in Jahn [79], Laumanns et al. [92] and Zitzler et al. [166] and references therein. Convergence properties of EA are investigated by modeling them by Markov chains ensuring positive transition probabilities. Convergence proofs have been shown for special EA operators only, an extensive study in this field is still due. We note that theoretical aspects, in particular for application problems, are still a remaining task in the current research. A systematic comparison of evolutionary multi-objective algorithms on several test functions was applied in Zitzler et al. [163] and an engineering relevant topology optimization problem is discussed in Deb et al. [29]. Overview papers to the subject are available in Hajela [61], Deb [26] and Horn [70]. From 1990, we cite Horst and Tuy [72] who discuss deterministic multi-objective algorithms, and quote a compendium of engineering relevant multi-objective problems and solution procedures in Eschenauer et al. [43]. However, research concerning about the application of multi-objective methods on demanding applications is still in its infancies.

Approximation Models Approximation models are a substitute to the functional coherence of the original mapping. The approximation model is a computationally cheap surrogate which can be used to produce system answers on unseen parameters in magnitudes of less time than the numerical solver needs. Strategically spoken, the information contained in each solver run is used to train a simulation tool which then produces target evaluations on unseen parameter combinations. This is a highly demanding task and often cannot be satisfyingly solved by conventional approximation tools like polynomial approximation (e.g. least squares methods).

There are several approaches how to set up an approximate model. The response surface method (RSM) is widely know and most often applied in the context of reliability analysis.

The objective of the RSM is to obtain a description of the influence of each variable and their possible combinations on the response of the system. This is achieved via a polynomial interpolation of input/output sample points. This regression then serves as a substitute for function evaluations.

Instead of a polynomial approximation, neural networks are recently proposed to be used as the approximation model. Neural networks are a form of multiprocessor computing systems, and consist of multiple simple, independent processing elements which are highly interconnected. These methods are intelligent techniques which are able to learn the governing rules of a system. The most distinguishing characteristic of neural networks is their ability to map highly nonlinear structures with very low computational complexity. Furthermore, they are exceptionally good at performing pattern recognition tasks and are used for identifying trends in data. Neural networks find application for problems in fields like regression analysis, data classification, time series prediction, industrial process control (e.g. control theory) as well as marketing and customer research, risk management, medicine and many more [90, 157]. Using approximation models for design optimization purposes is quite recent and has gained increased interest in the engineering community during the past decade.

Papadrakakis et al. [113,114] as well as Hurtado et al. [76,77] and other authors [16,52,107, 110] suggested a neural network approach as a surrogate model to substitute a numerical solver. The neural network approach has also found applications in reliability analysis, where an integral on very small support has to be approximated. Radial Basis Function (RBF) [85] networks proved to be most successful for these purposes. Giannakoglou [52] introduced the inexact pre-evaluation (IPE) method, working with reduced numerical precision in the early generations to accelerate the search process. However, progressive network models have for long time been the 'missing link' in optimizing applications that involve heavy computations. A novel aspect in our approach is using the Bayesian regularization [100] feature, designed to generalize especially well. We also investigate the adaptive neuro fuzzy inference system (ANFIS) [82] which is a network based on principles from fuzzy logic.

Numerical Simulation Since the second half of the 20th century, when computers became applicable for fundamental numerical use, numerical problems in structural mechanics have been explored by the aids of the finite element method (FEM), a field known as computational structure mechanics (CSD). Up to now, the finite element method is the only proven mathematical method of assessing the response of a complex structural system. It is though quite recent that computer processing power made it possible to provide simulations for fluid flow processes, a field known as computational fluid dynamics (CFD). The objective of CFD is to use computers to solve the previously intractable conservation equations for fluids in order to accurately simulate flows. This typically involves discretizing the problem in a finite set of elements, applying the conservation of mass, momentum, and energy to these elements, placing additional boundary conditions at the edges of the computational grids, and solving the resultant algebraic equations in an iterative fashion. The computations encountered here are much more demanding

and due to effects like turbulence, the computational requirements in CFD can be of orders of magnitude larger than in CSD problems. In context of fluid dynamics, it can be said that the development of numerical investigations are by far not finished, many open questions and problems remain in this field, like the simulation of turbulent flows or efficient computation in complex geometries. The finite volume method (FVM) has been established during the last decades to efficiently simulate fluid flow processes. A detailed description of the methods can be found in the standard literature as in Chattot [18], Lomax et al. [98], Ferziger and Perić [45], and Schäfer [130]. In this work, we used the flow solver FASTEST-2D [36,78]. It is based on a fully conservative finite volume method for solving the incompressible Navier-Stokes equations on a block-structured, cell-centered grid arrangement. The multigrid method is utilized for convergence acceleration.

1.3 Objectives and Scope of the Study

We consider evolutionary algorithms for several engineering relevant optimization problems. First, we apply EA's to optimization problems where only a single objective is considered. To improve EA performance, approximation models are employed. Another focus of this work is on the application of multi-objective optimization algorithms on shape optimization problems from fluid dynamics. To obtain an insight into design alternatives along the Pareto front, the application of evolutionary algorithms become inevitable. Conventional methods in this field like the weighted sum approach, are not very promising to be effective and efficient. We demonstrate the performance of progressive evolutionary multi-objective optimization algorithms on optimizing heat exchanger configurations. By this means, we apply the improved Strength Pareto Evolutionary Algorithm (SPEA2) and the elitist Non-dominated Sorting Algorithm (NSGA-II). Since EA techniques are tested on functions available in closed form, the number of evaluations usually do not play a major role. That is why heuristic methods are conceived as very computationally expensive as they require a large number of function evaluations. In 2-D flow computations, these function evaluations may involve several minutes to hours whereas for 3-D configurations the computations may take up to days of function evaluation time. Besides the long running function evaluation time, another major drawback involved with multi-objective computation is that a quantitative analysis is only available for analytic objective functions, where in this way, an a posteriori algorithm performance was easy to perform. So here, one has predominantly to rely on 'eye-sight' to what can be observed in objective space.

When fuzzy and neural computation methods usually find application in control theory (where pattern recognition tasks have to be solved), the contribution of this work, in particular, is to show the ability of the proposed networks in aspects of function approximation arising in fluid flow application context. We show the employment of two high-end neural network configurations to be used as substitute models for a demanding design optimization problem. The proposed networks will be seen to considerably improve the function approximation capabilities.

1.4 Overview of Contents

The arrangement of this work is as follows. Section two presents a framework introducing optimization methodology. Different methods are discussed and an engineering design process is defined in which the most important ingredients for optimizing in engineering context are considered. The last section of this chapter shows the application of evolutionary algorithms for problems from structural mechanics. In that, we see the working principles of EA and also highlight the importance of the proposed approach when confronted with a multi-objective problem.

Chapter three details the multi-objective optimization algorithms we employed and demonstrates the usefulness of these in context of optimizing two heat exchanger units due to several criteria.

In Chapter four, we first explain the neural network methodology and then discuss Bayesian regularization and adaptive neuro-fuzzy inference systems which are applied to a demanding fluid flow problem.

For each aspect of the engineering design optimization process, we give an overview of the available literature and highlight pros and cons and properties coming along with each of the available approaches.

Last, we summarize the results of this dissertation in Chapter five.

Chapter 2

Design Optimization Methodology

"A problem exists when there is a recognized disparity between the present and desired state. Solutions, in turn, are ways of allocating the available resources so as to reduce the disparity between the present and desired state."

Michalewicz and Fogel [104]

Optimization is often called mathematical programming, a term which has its roots in the early days of optimization where the word programming was not inextricably connotated with computer software. Optimization problems can be traced back to the calculus of variations and the work of Euler and Lagrange. Since then, most of the proposed algorithms are employed to find a parameter combination so as to satisfy the first necessary condition for an optimal point, expressed as $\nabla f(\mathbf{x}) = 0$.

Often, there will be parts of the design process that require human or unquantifiable judgment that is not suited for automation with an arbitrary optimization strategy. In context of engineering relevant optimization, these parts are quickly identified, for instance to choose between preferences of multiple objectives. This chapter is a survey of engineering relevant optimization methodology and serves as a basis for choosing the right solution routine for the problem at hand. We define an engineering design process by analyzing the most important ingredients of optimization strategies. Thereby, a number of conventional and also up-to-date algorithms are presented and discussed so as to highlight usability and application relevance. The general optimization problem can be formulated as

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^m} \quad & f_i(\mathbf{x}) : X \mapsto Y \quad && i = 1, \ldots, n \\ \text{subject to} \quad & c_i(\mathbf{x}) \geq 0, && i \in I_I \\ & c_i(\mathbf{x}) = 0, && i \in I_E, \end{aligned} \tag{2.1}$$

where $X \subset \mathbb{R}^m$ and $Y \subset \mathbb{R}^n$. The $c_i(\mathbf{x})$ are called constraints, the index sets $i \in I_I$ are inequality, $i \in I_E$ equality constraints. In our context, only linear inequality constraints

are considered to prevent the creation of invalid (e.g. negative) shapes. Determination of the extrema is a dual task; minimization and maximization problems can be transformed into each other by considering the negative of the objective. The wide scope of engineering optimization includes the design of optimal aircraft and aerospace structures, pumps, turbines, valves and heat transfer equipment for maximum efficiency, designs of mechanical components like linkages, cams, gears, machine tools as well as the optimal production planning, controlling and scheduling, and optimum design of control systems.

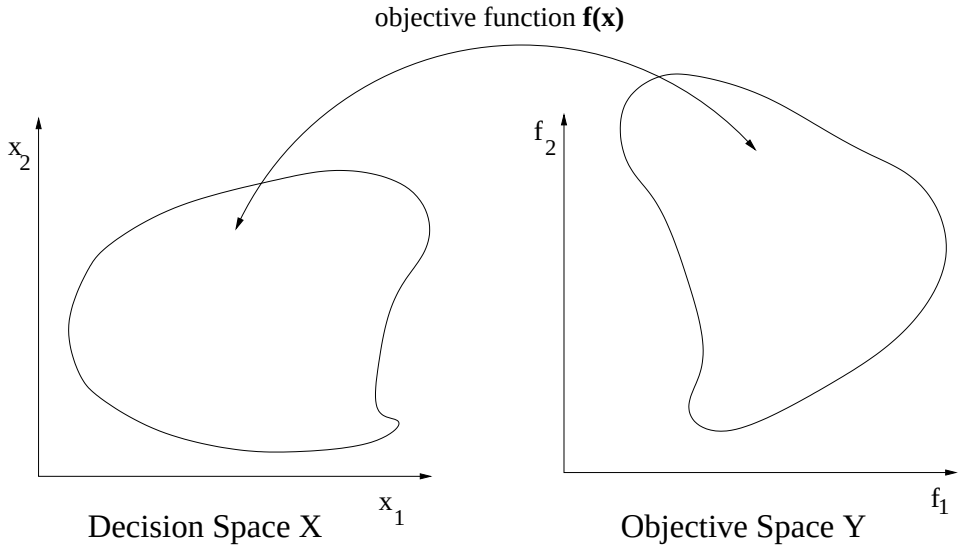


Figure 2.1: Mapping from decision space into objective space

In general, we have to deal with a mapping from $X \subset \mathbb{R}^m$ to $Y \subset \mathbb{R}^n$, $n > 1$ (cf. figure (2.1)). Obviously, what means 'optimizing' to a vector demands further specification. In fact, (2.1) is a multi-objective problem. This class requires special care to be taken of. In section 2.2.1 we first restrict our considerations to the case $n = 1$ and will discuss the multi-objective case in section 2.2.2. The following sections introduce to the engineering design process and concepts used therein.

2.1 The Engineering Design Process

Optimization consists of an analysis part which consists of the optimization model and optimization algorithm, and a synthesis part which can be seen as the development of new designs, thus involving the objective function and design parameters (cf. Andersson [1]). We define five major components in a design improvement process, these components are of immense importance in engineering applications and are independent fields of research where each component contributes significantly to the whole process.

Optimization Model - The Problem Nature The first task is to identify the governing optimization model. The challenge here is to translate the possible real world

or almost-real-world ('near-world') problem into an abstract setting. This process must not be underestimated, the identification of design variables, objective funtion, constraints, penalty parameters, problem scaling parameters, etc. have to be formulated and set up very carefully.

Transition Rule After setting up the optimization model, one has to decide for an optimization algorithm. Usually, there is a large number of possible algorithms for a specific problem class available where these methods overlap and can also be combined. A transition rule can either be deterministic, as in gradient descent, or of stochastic nature, as in evolutionary algorithms. The task is to find the right algorithm solving the actual problem.

The algorithms we propose in the later sections should be understood as in contrast to algorithms working with analytical objective functions or use an artificial setup on which the optimization process takes place.

Design Evaluation This part of the process concerns the evaluation of design proposals. The optimization process requires an objective function which is here a numerical solution of a physical process. The design evaluation is an important part since each evaluation can take up several minutes to hours of computing time. Thus, efficient numerical methods are necessary. As we shall see later, the immanent characteristics of the objective function strongly influence the optimization model and algorithm.

Design Development Design development is about the generation of new and hopefully better designs. Numerical evaluation requires a discretization defining a grid which predominantly determines the precision and convergence speed of the approximation. The parametrization of the geometry is not straightforward and an efficient grid is mandatory for the computing process. Moreover, the parametrization of the geometry essentially defines the search space, hence diverse geometry parametrizations will most often yield different optima.

Appropriate Design Strategy This will sum up and merge the gathered insights. The optimization model, optimization algorithm and the objective function properties will be seen to influence the whole problem setting.

We discuss each component exclusively in the proceeding sections.

2.2 Problem Nature

Optimization takes place in a large number of disciplines and up to today, there are a lot of optimization algorithms proposed. Optimization algorithms can be classified according to the problem they solve: A linear programming method finds application to optimization problems in which the objective function as well as all constraints are linear functions. In nonlinear programming algorithms, at least one of the constraints or the objective function appears to be nonlinear. Naturally, nonlinear problems arise predominantly in physical

science and engineering whereas linear problems are more common in management science and operations research. In most cases, the class of problem encountered will dictate the type of solution procedures to be adopted in solving the problem. We can classify optimization problems according to the following.

Constraints An important distinction is if the optimization problem is subjected to constraints and if so, of which kind they are. In numerical optimization algorithms, it is the most distinguishing feature of algorithms. Most popular methods for unconstrained problems are line search algorithms like Newton's method or gradient descent algorithms. In constrained problems, the optimization algorithm is only meaningful if the (most often inequality) constraints are fulfilled. The Karush-Kuhn-Tucker conditions describe the necessary conditions that have to be met for solving constrained problems. Trust region techniques like sequentiell quadratic programming are algorithms tailored to meet these conditions.

Number of Objectives Multi-objectivity is an important property which has only recently found appreciation in the optimization community. This work especially emphasizes on the multi-objective nature of design problems on which we will detail methods in the subsequent sections and provide computations in the proceeding chapter.

Global and Local Search Most optimization algorithms are designed to find a local minimum, thus, they depend on the iteration starting point. Each optimum then corresponds to one optimal design in physical space. Naturally, the user wants to learn more about a design and is interested to investigate each local optimum. This is because the optimal designs may possess certain extra features, e.g. stability properties. An example of this is given, e.g., in Haase [55]. Certainly, the global optimum is of greatest interest and should also be found, especially since we do not know function properties a priori.

In this context, it is desired to apply evolutionary algorithms in such a way that the search radius in the first generations is sufficiently large so that the whole search domain is covered and thus global information is gathered.

Availability of the Gradient Optimization is simple if the gradient and Hessian matrix are available in analytic form. This is most often not the case and thus, an approximation to the Hessian and the gradient have to be made. Numerical optimization deals with optimization problems where the derivatives are approximated by numerical methods.

Nature of Design Variables In a parametric problem, the task is to find values for the design variables such that the objective function becomes optimal in a certain sense. When the design variables are functions of another parameter, the problem is called dynamic optimization problem.

Nature of the Equations involved The functional behavior of the objective can not only be linear or nonlinear. There is large variety of functional behavior that can be classified, e.g. as geometric, dynamic, stochastic or multivariate.

Discrete or Real Valued Design Variables In parameter optimization problems, the design variables are usually real values. There are also optimization problems where the desgin variables cannot be stated as fractions, this defines the class of discrete or, integer programming problems. In topology optimization, the design variables either contain a $'0'$ or a $'1'$ to indicate if a design part is active or not.

Deterministic or Stochastic Nature This distinction is especially interesting for many engineering applications. The design can be of stochastic nature, typical examples [120] in engineering practice are sea waves, earthquake ground motion, road roughness, imperfection of shells, fluctuating properties in random media, etc. The proper mathematical concept for this class are stochastic differential equations. These are differential equations which have a random variable as an unknown. The optimization aims at determining the expected performance of the model.

In this work we deal with a deterministic nature of design variables, each set of design variables corresponds to an exactly known output.

2.2.1 Single Objective Optimization

We first discuss some properties of objective functions that only involve a single objective. Section 2.2.1.1 states some basic optimization principles which already serve as solvers for problems available in analytic form. Section 2.2.1.2 considers the case for constrained problems, in these, optimality conditions are less easy to formulate. Bäck [7] and Nocedal and Wrigth [109] are basic literature in this aspect and the following sections will stay close to their methodology. We first formalize what is meant by a minimum. To these ends, we need

Definition 2.2.1 *A neighbourhood (equivalent: an n-dimensional ball) $\mathcal{B}_\Delta$ around $\mathbf{x} \in \mathbb{R}^m$ is a set of values*

$$\mathcal{B}_\Delta(\mathbf{x}) := \{\mathbf{y} \in \mathbb{R}^m |\ ||\mathbf{y} - \mathbf{x}|| \leq \Delta\},$$

with $\Delta > 0$ the neighbourhood radius and $||.||$ an appropriate norm.

The next definitions are of significant relevance when discussing optimization procedures.

Definition 2.2.2 (Local Minimum) *A local minimum is a point $\mathbf{x}^*$ such that $f(\mathbf{x}^*) \leq f(\mathbf{x})$ in a neighbourhood $\mathcal{B}_\Delta$ around $\mathbf{x}^*$.*

Definition 2.2.3 (Global Minimum) *A global minimum is a point $\mathbf{x}^*$ such that $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in X \subset \mathbb{R}^m$ with X the domain of the design variables.*

Usually, optimization algorithms are local procedures, i.e. the optimum they find depends on the starting point of the iteration. This is highly unintentional for practical applications since in engineering design it is of great importance i) to find the global minimum, and ii) *simultaneously* (i.e. in one optimization run) find all local minima. The upcoming sections discuss local as well as global approaches to which the later chapters provide computational examples and also a comparison of methods.

2.2.1.1 Optimality Conditions I: Unconstrained Optimization

The unconstrained optimization problem is simply

$$\min_{\mathbf{x}\in\mathbb{R}^m} f(\mathbf{x}) \tag{2.2}$$

To be meaningful, the function f has to be bounded away from $-\infty$. In the present and the next section, we assume this and also that f is sufficiently smooth to fulfill the discussed optimality conditions. By smoothness of $f : X \subset \mathbb{R}^m \mapsto \mathbb{R}^n$ with degree p we understand

Definition 2.2.4

$$C^p(X) := \{f(\mathbf{x}) \mid f(\mathbf{x}) \textit{ is } p \textit{ times differentiable for all } \mathbf{x} \in X \textit{ and } f^{(p)} \textit{ is continuous}\}.$$

The next definition is also of key importance in optimization.

Definition 2.2.5 (Convex) *A set $X \subset \mathbb{R}^m$ is convex if for any two $\mathbf{x} \in X$ and $\mathbf{y} \in X$ we have*

$$\alpha\mathbf{x} + (1-\alpha)\mathbf{y} \in X \quad \textit{for all } \alpha \in [0,1].$$

f is a convex function if its domain X is convex and if for any two points $\mathbf{x} \in X$ and $\mathbf{y} \in X$ it holds

$$f(\alpha\mathbf{x} + (1-\alpha)\mathbf{y}) \leq \alpha f(\mathbf{x}) + (1-\alpha)f(\mathbf{y}), \quad \textit{for all } \alpha \in [0,1].$$

For a convex function, it is guaranteed that, if an optimization algorithm converges, it converges to its minimum. For unconstrained optimization, the concept of convexity allows that local and global optimization coincides. If convexity is unavailable, optimization routines will usually get stuck in local minima. Convex functions are considered as unproblematic from optimization point of view.

The following are statements from basic calculus. Unfortunately, optimization methods often are designed such that these necessary conditions are fulfilled. As soon as it comes to applications, the very strong assumptions in the following theorems are not valid anymore.

Theorem 2.2.6 (Necessary Condition of 1st Order) *Let $\mathcal{B}$ be a neighborhood of $\mathbf{x}^*$ and $\mathbf{x}^*$ be a local extremum of a function $f \in C^1(\mathcal{B})$, then $\nabla f(\mathbf{x}^*) = 0$ (i.e. $\mathbf{x}^*$ is a stationary point of f).*

Theorem 2.2.7 (Necessary Condition of 2nd Order) *Let $\mathbf{x}^*$ be a local minimum of a function $f \in C^2(\mathcal{B})$. Then $\nabla f(\mathbf{x}^*) = 0$ and $\nabla^2 f(\mathbf{x}^*)$ is positive semidefinite.*

Theorem 2.2.8 (Sufficient Conditions of 2nd order) *Let the Hessian $\nabla^2 f$ be continuous in a neighborhood of $\mathbf{x}^*$ and $\nabla f(\mathbf{x}^*) = 0$ and $\nabla^2 f(\mathbf{x}^*)$ positive definite, then it follows that $\mathbf{x}^*$ is a local minimum of f. Furthermore, for $f : X \subset \mathbb{R}^m \mapsto \mathbb{R}$ is convex, each local minimum $\mathbf{x}^*$ is also a global minimum of f. If, additionally, $f \in C^1(X)$, then each stationary point is a global minimum.*

2.2.1.2 Optimality Conditions II: Constrained Optimization

This section reviews theoretical aspects when optimizing constrained optimization problems. We consider (2.1), this time with $n = 1$ and the constraints

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^m} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & c_i(\mathbf{x}) \geq 0, \quad i \in I_I \\ & c_i(\mathbf{x}) = 0, \quad i \in I_E. \end{aligned} \tag{2.3}$$

The constraints change the essential nature of the problem and any algorithm now needs to take special care about them. The Lagrange function $\mathcal{L}$ is defined and used to incorporate the objective function and the constraints into one function,

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) \;:=\; f(\mathbf{x}) - \sum_{i\in I_E\cup I_I} \lambda_i c_i(\mathbf{x}), \quad \lambda_i > 0,$$

where f as well as c_i for all $i \in I_E \cup I_I$ have to be in C^1, i.e. they need to be continuously differentiable. The λ_i are called Lagrange parameters or Lagrange multipliers. An essential property of the Lagrangian $\mathcal{L}$ is that a saddle point is simultaneously a local minimum of the objective function (though this assumes existence of minima). It is now of particular interest to formulate the necessary conditions for a point $\mathbf{x}^*$ be an optimum of (2.3).

By this means, we know that when for unconstrained minimization problems necessary and sufficient conditions are expressed by their derivatives and most often broadly well known, the corresponding does not hold for constrained problems. Again, optimality conditions are of practical importance since optimization algorithms are developed to fit these conditions. We will later present an algorithm which directly makes use of the Karush-Kuhn-Tucker conditions. To these ends, we first define the set of active indices. For a feasible $\mathbf{x}$,

$$I_A(\mathbf{x}) := I_E \cup \{i \in I_I : c_i(\mathbf{x}) = 0\}.$$

This definition also has important consequences in theoretical investigations. In context of optimization algorithms, the concept of working with active indices means that no optimization progress needs to be made while inequality constraints are not at its boundary of domain. It means that optimization can be done by ignoring inequality conditions, they are treated as they were equality constraints. Moreover, the concept of working with active indices ensure that the Lagrangian vector will be unique (see below). We have

Definition 2.2.9 (LICQ Conditions) *The linear independence constraint qualification (LICQ) holds if, for a given point $\mathbf{x}^*$ and the corresponding active set $I_A(\mathbf{x}^*)$, the set of active constraint gradients $\{\nabla c_i(\mathbf{x}^*), i \in I_A(\mathbf{x}^*)\}$ is linearly independent.*

The set $K_2(\boldsymbol{\lambda}^*)$ in the following definition is a cone used to ensure that the Lagrange parameters stay positive in a region around the minimum.

Definition 2.2.10 *Given the feasible point* $\mathbf{x}^*$ *and the active constraint set* $I_A(\mathbf{x}^*)$*, we define the cone*

$$K_1(\mathbf{x}^*) := \left\{\alpha\,\mathbf{d} \mid \mathbf{d}^T\nabla c_i(\mathbf{x}^*) = 0,\ i \in I_E,\ \mathbf{d}^T\nabla c_i(\mathbf{x}^*) \geq 0,\ i \in I_A(\mathbf{x}^*) \cap I_I\right\}, \quad \alpha > 0,$$

and the subset of K_1

$$K_2(\boldsymbol{\lambda}^*) := \left\{\mathbf{w} \in K_1(\mathbf{x}^*) : \nabla c_i(\mathbf{x}^*)^T\mathbf{w} = 0,\ i \in I_A(\mathbf{x}^*) \cap I_I \text{ with } \boldsymbol{\lambda}_i^* = 0\right\}.$$

Theorem 2.2.11 (Necessary Conditions) *Suppose that* $\mathbf{x}^*$ *is a local solution to (2.3) and that the LICQ holds at* $\mathbf{x}^*$*. Then there exists a Lagrange multiplier vector* $\boldsymbol{\lambda}^*$*, such that the following first order conditions are satisfied at* $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$

$$\begin{aligned} \nabla_x\mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*) &= 0, \\ c_i(\mathbf{x}^*) &= 0, i \in I_G \\ c_i(\mathbf{x}^*) &\geq 0, i \in I_I \\ \lambda_i^* &\geq 0, i \in I_I \\ \lambda_i^* c_i(\mathbf{x}^*) &= 0, i \in I_E \cup I_I. \end{aligned} \tag{2.4}$$

These conditions are known as Karush-Kuhn-Tucker conditions. If these conditions are met, then the second order necessary condition is

$$\mathbf{w}^T\,\nabla_{xx}\mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*)\,\mathbf{w} \geq 0, \quad \textit{for all } \mathbf{w} \in K_2(\boldsymbol{\lambda}^*).$$

The KKT are necessary conditions for non-convex problems and necessary and sufficient for convex problems. We can rewrite (2.4) as

$$\nabla_x\mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*) \;=\; \nabla f(\mathbf{x}^*) - \sum_{i \in I_A(\mathbf{x}^*)} \lambda_i^* \nabla c_i(\mathbf{x}^*).$$

It should be noted that the derivation of Theorem 2.2.11 is quite mathematically involved. For completeness, we furthermore state

Theorem 2.2.12 (Sufficient Conditions of 2nd order) *Suppose that for a feasible point* $\mathbf{x}^* \in \mathbb{R}^m$ *there is a Lagrange multiplier vector* $\boldsymbol{\lambda}^*$ *such that the Karush-Kuhn-Tucker conditions are satisfied. If furthermore holds that*

$$\mathbf{w}^T\nabla^2_{xx}L(\mathbf{x}^*, \boldsymbol{\lambda}^*)\mathbf{w} \;>\; 0 \quad \textit{for all } \mathbf{w} \in K_2(\boldsymbol{\lambda}^*) \textit{ with } ||\mathbf{w}|| \neq 0,$$

then $\mathbf{x}^*$ *is a local solution of (2.3).*

2.2.2 Multi-Objective Optimization

Optimizing multiple objectives is quite a recent task in the context of mechanical engineering applications. Originally, these concepts were introduced by the 20th century economists Edgeworth [38] and Pareto [115]. In multi-objective optimization problems, a

decision between multiple, but equally important objectives is sought. We consider the full constrained multi-objective problem (2.1) with $n > 1$:

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^m} \quad & f_i(\mathbf{x}) \qquad i = 1, \ldots, n \\ \text{subject to} \quad & c_i(\mathbf{x}) \geq 0, \quad i \in I_I \\ & c_i(\mathbf{x}) = 0, \quad i \in I_E \end{aligned}$$

We always assume that in the presence of more than one objective, they are concurrenting (conflicting), otherwise the discussion would be trivial (the optimization problem would then naturally be a single objective problem). Here, we aim at determining an optimal vector $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_n(\mathbf{x}))$ which components are to be maximized or minimized according to certain criteria. As Horn [70] elucidates, multi-criteria decision making involves two types of problem difficulty: search and multi-criteria decision making. These are not independent tasks, consider, for instance, simulated annealing which is a search algorithm but does not employ any decision making features. On the other hand, the multi-objective literature most often assumes small search spaces and emphasize on the decision process. It is common that the space to be searched can be too complex (multi-modal, nonlinear, etc.) to be solved by linear programming or a local or gradient method. If the objectives are conflicting, then the search space is strictly instead of partially ordered [53].

We consider as important to account for the *no free lunch theorem* [108, 158], according to which there cannot be a single optimization algorithm which would be best for solving all types of optimization problems. It thus makes sense to concentrate on one class of optimization problems such as linear, quadratic, convex, or multi-modal programming problems etc. It is then the task to find the best algorithm for solving in the particular class of problem.

The Pareto-Dominance Concept

We first introduce some basic concepts necessary to understand the nature of the problem. We define the *decision space* $\mathbb{R}^m$ and the *objective space* $\mathbb{R}^n$ (cf. figure (2.1)) from $\mathbf{f} : \mathbb{R}^m \mapsto \mathbb{R}^n$, $n > 1$. In the above mapping, n represents the number of conflicting objectives. A parameter vector from decision space $\mathbf{x} = (x_1, \ldots, x_m) \in \mathbb{R}^m$ is exposed to fitness evaluation by a fitness function $\mathbf{f}$ such that $\mathbf{x}$ is evaluated in n objectives, denoted as

$$\mathbf{f}(\mathbf{x}) \;=\; (f_1(\mathbf{x}), \ldots, f_n(\mathbf{x})) \;=\; (f_1, \ldots, f_n) \in \mathbb{R}^n.$$

In the case of $n = 1$, the goal is to minimize a single objective. In such a single-objective problem, a solution $\mathbf{x}_1 \in \mathbb{R}^m$ is better than another solution $\mathbf{x}_2 \in \mathbb{R}^m$ if $f(\mathbf{x}_1) < f(\mathbf{x}_2)$, meaning that there is a well defined optimum in the objective space. For a vector-valued evaluation function $\mathbf{f}$ with $n > 1$, the situation of comparing two solutions becomes more involved. Again, it is important to note that either objective is equally important. We begin our discussion with key concepts in this field. Formulations presented here correspond to the mathematical illustration most widespread in multi-objective optimization

literature, see, e.g. Jahn [79], Steuer et al. [146] and Ringuest [126]. First, two basic definitions concerning the ordering of sets are given.

Definition 2.2.13 (Partial Order) *Let X be a set. Each nonempty subspace of the product space $X \times X$ is called a binary relation on X. A binary relation qualifies for an ordering relationship if it is at least transitive, i.e.*

$$x \leq y,\ y \leq z \Longrightarrow x \leq z \text{ (for all } x, y, z \in X),$$

and fulfills one further of the following properties (valid for all $x, y, z \in X$).

i) $x \leq x$ *(reflexive)*
ii) $x \leq y \Longrightarrow y \leq x$ *(symmetric)*
iii) $x \leq y \Longrightarrow y \not\leq x$ *(asymmetric)*
iv) $x \leq y,\ y \leq x \Longrightarrow x = y$ *(antisymmetric)*

A binary relation is called a partial ordering on X if it is reflexive, transitive and antisymmetric. A strict partial order is asymmetric and transitive. A linear space equipped with a (strict) partial ordering is called a (strictly) partially ordered space.

Definition 2.2.14 (Pareto Dominance) *In general, the vector $\mathbf{x} = (x_1, \ldots, x_n)$ is said to dominate the vector $\mathbf{y} = (y_1, \ldots, y_n)$ in the Pareto sense iff*

$$\mathbf{x} <_P \mathbf{y} \ :\Longleftrightarrow\ (\forall i \in I:\ x_i \leq y_i)\ \wedge\ (\exists i \in I : x_i < y_i)\,,$$

where $I = 1, \ldots, n$. The formulation is equivalent for maximizing objectives, i.e. switching the inequality symbols to $>_p$, $>$, and $\geq$.

This leads to the essential concept of

Definition and Theorem 2.2.15 (Pareto Optimality) *Let $f : X \subset \mathbb{R}^m \mapsto Y \subset \mathbb{R}^n$ be a given vector-valued function. $\mathbf{x}^* \in X$ is called a Pareto optimal solution if $\mathbf{f}(\mathbf{x}^*)$ is a minimum element of the image set $\mathbf{f}(X)$, i.e. there exists no $x \in X$ such that $f(\mathbf{x})$ dominates $\mathbf{f}(\mathbf{x}^*)$. The expression 'Pareto optimal' is taken to mean with respect to the entire decision variable space unless otherwise stated.*

$\mathbf{f}(\mathbf{x}^*)$ is also called non-dominated, non-inferior, admissable or, efficient solution. The nominal names depend on the scientific field in which the Pareto concept is applied.

The union of non-dominated solutions is called the Pareto set, and we will denote its image in objective space as Pareto front. The set of Pareto-optimal solutions consists of all non-dominated points in the decision variable space. The Pareto front is the image of these vectors in the objective space.

Definition 2.2.16 (Pareto Optimal Set) *For a given multi-objective problem, the Pareto optimal set $\mathscr{PS}^*$ is defined as*

$$\mathscr{PS}^* := \{x \in X \mid \nexists\ x' \in X:\ f(x') <_P f(x)\}$$

Definition 2.2.17 (Pareto Front) *For a given multi-objective problem and Pareto optimal set $\mathscr{PS}^*$, the Pareto front $\mathscr{P}^*$ is defined as*

$$\mathscr{P}^* := \{\mathbf{f}(\mathbf{x}) \mid \mathbf{x} \in \mathscr{PS}^*\}$$

In other words, the solution $\mathbf{x}^*$ is no worse than any other solution in any objective, but strictly better in at least one objective. The other way around, $\mathbf{x}^*$ is optimal in the sense that it cannot be improved in any objective without causing a degradation in at least one other objective. This set of non-dominated points constitutes a strict partial order relationship and $(Y, <_P)$ is a strictly partially ordered space. Zitzler et al. [167] and Veldhuizen [154] discuss problems concerning the assessment of solution sets in relation to multi-objective algorithms. Jahn [79] is a contemporary work on vector optimization problems. To illustrate, figure (2.2) presents several solutions where both objectives f_1

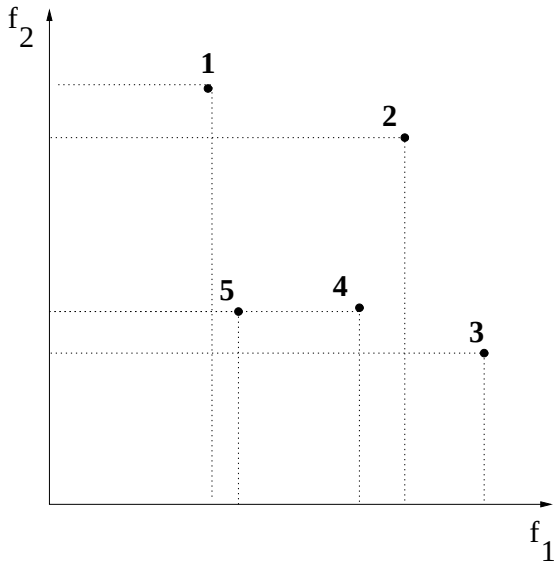


Figure 2.2: Emerging Pareto-front. Both objectives (f_1 and f_2) are to be maximized.

and f_2 are to be maximized. Solution 4 is dominating solution 5 since solution 4 reaches a higher maximization result in objective 1. Though, both solutions are undistinguishable according to objective 2. So we can say that solution 4 dominates solution 5. On the other hand, solution 1, for instance, is better than solution 2 in maximizing objective 2 but much less successful in maximizing objective 1. So one cannot say if solution 1 or solution 2 is preferable, the preference about choosing one of these solutions requires subjective information. We say that solutions 1 and 2 are non-dominated to each other since we do not know *a priori* if either one of them is better than the other. Furthermore, solution 2 can be seen as clearly dominating solutions 4 and 5. The Pareto-optimal solutions are the points 1, 2, 3 in the objective space (figure (2.2)), on the 'border' of the output parameter cloud. Figure (2.3) is a computational example in which maximum stress and volume of a tensile bar are to be minimized and clearly concurrent for an optimal solution. The Pareto front represents optimal solutions for which we cannot state that any two of them should be given preferention. So we are not able to decide for two

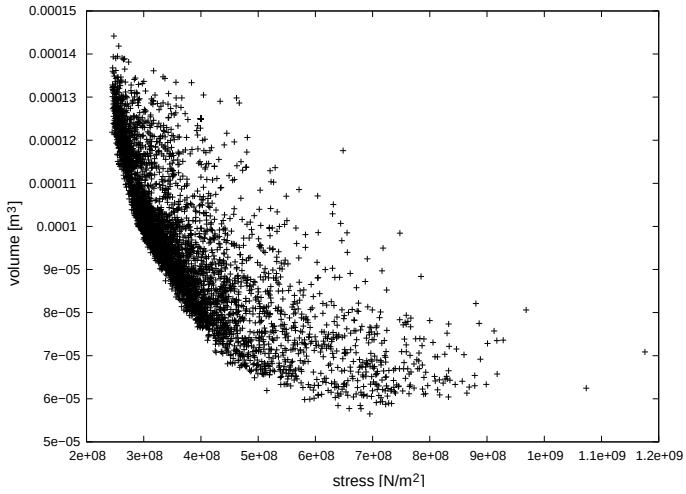


Figure 2.3: Pareto Front according to a numerical example (cf. section 2.3.1.5), [135]

arbitrary points on the Pareto front to which we may give higher appreciation. This is a characteristic feature of optimizing multiple objectives, the incomparableness of solutions. Now, a further information about the preference of equally worth solutions is necessary. This preference information cannot be included in the optimization process since this is the actual task of the user. Formally, this can be expressed as the fact that no usual unary relation can be defined [154,167] (e.g. a one dimensional quality measure of solution sets). The binary relation here qualifies for a strict partial ordering, we note that this has far-reaching consequences for the solution sets.

The above means that the Pareto set has the outstanding property that it entails equivalent designs and every decision along the Pareto front has to be accompanied by either subjective judgement or a further problem dimension. Pareto-front approximation is a very recent issue in the EA community and several methods have been suggested to render successful approximation. An important implication provided by the knowledge of the Pareto front is that the engineer is enabled to choose the best compromise solution according to the user's preferences. Furthermore, the design space is reduced to efficient solutions.

2.3 Transition rule

The transition rule refers to the way an optimization algorithm approaches a minimum. Optimization strategies set up rules which determine how the iterates $\mathbf{x}_k$ are driven to an optimal state $\mathbf{x}^*$ according to a prescribed optimality condition. In each iteration stage k, an objective function $\mathbf{f}$ is evaluated. This can be any measure which indicates the performance of an underlying system. In this work, the objective function is the solution of an approximation algorithm to the solution of a differential equation modelling structural or flow phenomena. This function value $\mathbf{f}(\mathbf{x}_k)$ is then used to determine a new iterate $\mathbf{x}_{k+1}$. Thus, optimization is an iterative process continued until convergence, i.e.

no more substantial achievement in the objective can be made anymore. Usually, the algorithm aborts due to an iteration number limit. The way how the new iterate $\mathbf{x}_{k+1}$ is extracted from state k distinguishes optimization algorithms. In general, algorithms can be categorized as we illustrate in the following.

2.3.1 Optimization Methods for Single Objective Problems

There is a huge number of optimization algorithms available (e.g. [11, 109]) where in the following we discuss a few of which are relevant for our purposes. Figure (2.4) indicates the algorithmic routine which underlies all conventional optimization algorithms employed to improve a criterion depending on design variables.

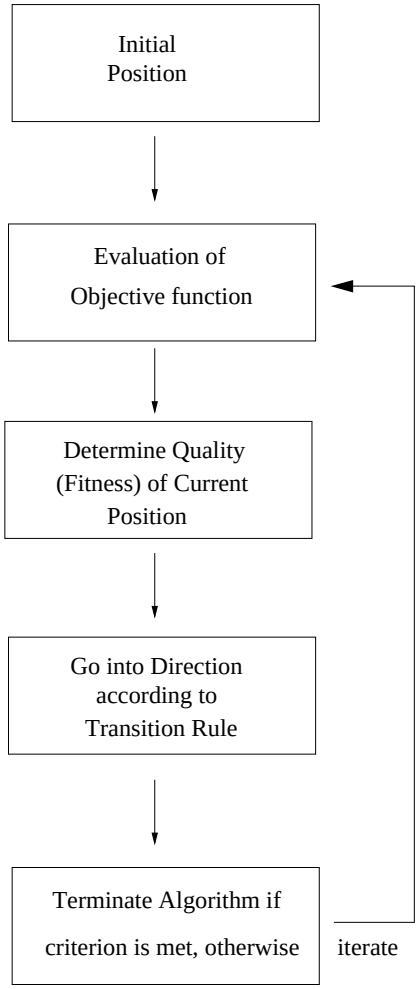


Figure 2.4: Flowchart Optimization Process

In general, one can distinguish optimization techniques in the categories calculus based, enumerative, and random [8]. Calculus based methods usually require the existence of derivatives and a certain degree of smoothness. Realistic problems do not possess derivates

that can be expressed in analytic form, hence calculus methods are difficult to apply in shape optimization. Enumerative methods are search schemes working on function evaluations. They can be applied to optimization problems where the search space is small, i.e. finite. Again, most realistic optimization problems have continuous decision variables and thus a countless number of possible design proposals. Random search algorithms are of special interest in this work and will be discussed in detail in the course of this work. The mentioned categories are however not exclusive but rather overlap and can also be combined. So one can evaluate the objective function several times at a deterministically or randomly determined point in the search space, using the information to set up an approximate model and then apply a gradient descent algorithm on this model (SQP methodology), an approach also known as derivative-free methodology. The drawback of derivative-free methods is that we cannot prove that we have found the actual optima. Comparative studies of different types of non-derivative methods can be found in Hajela [61].

Similar to search methods only working with a finite number of function evaluations, a very early development is the simplex method by Spendley [142] and Nelder and Mead [106]. A further development is Box' complex method [13]. Other approaches are polyhedron search methods, all employing a deterministic strategy to a certain number of function evaluations in the objective space. Algorithms especially designed for engineering applications can be found in, e.g., Brousse et al. [14], Baier et al. [8], and for shape optimization for flow problems in Pironneau [118] and Pironneau et al. [119]. This section outlines several classic optimization procedures which are detailed in Nocedal and Wrigth [109], Peressini et al. [10,117], and Bonnans et al. [11]. For realistic engineering applications the following optimization methods and strategies are often used.

Line search algorithms These are best known and well investigated from a mathematical point of view, such that convergence and convergence speed are often known for model problems. Most often, there are theorems stating assumptions under which convergence is guaranteed and also how fast the convergence is. In real world applications, these assumptions are nearly never satisfied (e.g. convexity). Line search algorithms are usually local procedures and are tested and designed for given analytical functions. The best known class are gradient based methods is the conjugate gradient approach, a search algorithm which moves towards the steepest descent direction, thereby 'conjugating' directions by selecting orthogonal search paths.

Approximation models After setting up a substitute to the function, many design methodologies from gradient based methods become applicable. The drawback is that a gradient obtained from an approximation model does not correctly map the functional behavior and so influences the efficiency of the method. On the other hand, approximation models are also able to tackle problems where no gradient is available, thus they are suitable for complex numerical simulations.

Stochastic methods These use stochastic operators instead of deterministic rules in order to direct solutions. As the name suggests, there is a lack of mathematical foundation for methods in this class. In case of evolutionary algorithms, convergence

proofs are only available for a few exemplary objective functions. For objective functions from a finite element or finite volume model, heuristics can be considered superior to deterministic techniques - *if* deterministic methods like gradient descent are applicable at all, which is most often not the case as we shall see soon.

Penalty function approaches Here, a penalty function measures the extent to which the objective is violated in terms of the constraints. This value is then added to the objective function. In this way, the optimization process is biased towards regions where constraints are satisfied.

Direct methods In direct methods, the optimization model is not changed as in the penalty function approach. The method of feasible directions is a popular representative of this class. Further methods are evolutionary strategies and also SQP methods which will be discussed in the ongoing sections. A witty presentation of direct methods is given by Trosset [152].

Methods based on optimality conditions Here, certain conditions which the design has to fulfill are formulated and the optimization process is then directed onto meeting the design requirements. The optimality conditions are derived from physical properties of the model.

The next section presents some basic optimization procedures taken from Nocedal and Wright [109]. Newton's method will not find application in engineering design although the principles underlying this method are important and find applications in many other algorithms as we shall see later.

2.3.1.1 Newton's method

Newton's method belongs to the class of gradient based methods. It is used to solve unconstrained nonlinear systems where the task is to

$$\min_{\mathbf{x}\in\mathbb{R}^m} \; f(x_1,\ldots,x_m).$$

The origin of this method was to the need to find a way to determine the stationary point

$$\nabla f(\mathbf{x}) = 0$$

which is done by setting up a quadratic model of f in a neighbourhood of its current iterate via the second order Taylor expansion of f around $\mathbf{x}_k$

$$f(\mathbf{x}_k+\mathbf{p}) \approx f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T\mathbf{p} + \frac{1}{2}\mathbf{p}^T\nabla^2 f(\mathbf{x}_k)\mathbf{p} \;=:\; m_k(\mathbf{p}), \tag{2.5}$$

where $\mathbf{p}$ is a search direction. By differentiating with respect to $\mathbf{p}$, the minimum necessarily fulfills

$$\frac{\partial}{\partial\mathbf{p}} f(\mathbf{x}_k+\mathbf{p}) = \nabla f(\mathbf{x}_k) + \nabla^2 f(\mathbf{x}_k)\mathbf{p} = 0. \tag{2.6}$$

This can also be understood as the Newton direction $\mathbf{p}$,

$$\mathbf{p} := -[\nabla^2 f(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k).$$

Now, by choosing $\mathbf{p}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ in (2.6) and iterating, we have

$$\nabla f(\mathbf{x}_k) + \nabla^2 f(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k) = 0, \qquad k = 0, 1, \ldots$$

and obtain the new iterate $\mathbf{x}_{k+1}$ as

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\nabla^2 f(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k).$$

The quadratic model m_k in (2.5) is often written as

$$f(\mathbf{x}_{k+1}) \approx f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T (\mathbf{x}_{k+1} - \mathbf{x}_k) + \frac{1}{2} (\mathbf{x}_{k+1} - \mathbf{x}_k)^T \nabla^2 f(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k).$$

The Newton method is of second order convergence speed and thus considered to be very fast, the best performance of the algorithm is achieved near the minimum. If the starting point for the iterative process is not close to the true solution, the Newton iterative process might diverge.

There exist a large number of varieties of Newton's method all using the explicit computation of the gradient and the Hessian. This is the main drawback with the Newton method - explicit computation of this matrix can be very expensive, if at all possible. That can be circumvented by using an approximation to the Hessian, known as a Quasi-Newton method, often of same convergence speed as the original method. Quasi-Newton uses an approximation which is updated after each step to take account of the additional knowledge gained during the step. The update makes use of the fact that changes in the gradient provide information about the second derivative of f along the search direction. The most popular quasi-Newton method is the BFGS algorithm, named after their founders Broyden [15], Fletcher [49], Goldfarb [54], and Shanno [141].

2.3.1.2 Gradient Descent

The idea in gradient descent algorithms is to search in a descent direction. The most obvious choice for a descent step (steepest descent) is $\mathbf{p} = -\nabla f(\mathbf{x})$ where the function value decreases at the fastest rate. The major drawback with gradient descent is that i) the gradient has to be known explicitly and ii) moving into a descent direction is a local feature. The gradient descent idea also influenced other methods.

Gradient descent is a classic line search algorithm. This is because, once the descent direction is set, it has to be determined how far the tentative step would reach by going into this direction. From a more formal point of view, gradient descent methods are based on the successive minimization in one-dimensional subspaces. A corresponding algorithm reads

Initial solution $\mathbf{x}_0$, $k = 0$

```
while  ||∇f(x_k)|| < ε
  p_k = −∇f(x_k)
  Determine α_k from min_{α∈ℝ} f(x_k + α p_k)
  x_{k+1} = x_k + α_k p_k
  k = k + 1
end
```

In each of the above iteration steps, one has to solve the one-dimensional problem

$$\min_{\alpha \in \mathbb{R}} \; f(\mathbf{x}_k + \alpha \mathbf{p}_k)$$

meaning to solve the nonlinear equation

$$0 = \frac{\partial}{\partial \alpha} f(\mathbf{x}_k + \alpha \mathbf{p}_k) = \nabla f(\mathbf{x}_k + \alpha \mathbf{p}_k)^T \; \mathbf{p}_k$$

for α. This can be done, e.g., by the Newton method with a series of values $\alpha_\nu,\ \nu = 1, 2, \ldots$

$$\nabla f(\mathbf{x}_k + \alpha_\nu \mathbf{p}_k)^T \; \mathbf{p}_k + (\alpha_{\nu+1} - \alpha_\nu) \; \mathbf{p}_k^T \nabla^2 f(\mathbf{x}_k + \alpha_\nu \mathbf{p}_k) \; \mathbf{p}_k = 0.$$

Gradient descent algorithms differ in how solving the one-dimensional minimization problem and also in determining new directions p_k, resulting in, for instance, the conjugate gradient procedure. This variant works on search directions which are a linear combination of the preceding search and the current gradient direction. It ensures that the angle between consecutive search steps is not too small, hence the algorithm converges faster.

2.3.1.3 Sequential Quadratic Programming

Sequentiell quadratic programming is a quite recent and one of the most powerful optimization techniques for unconstrained problems so far, designed to approximate minima for constrained optimization problems. Thereby, since the method is equivalent to Newton's method, its theoretical basis can be explored from results related to solutions of a set of nonlinear equations using Newton's method. On the other hand, SQP, as the name suggests, is based on a successive computing of a quadratic approximation and thus used in practical constraint optimization problems.

The most interesting issue about SQP, on contrary to Newton's or gradient descent methods is the fact that constraints are included in the optimization model. The method is constructed to fulfill the necessary conditions for constrained problems, i.e. the Karush-Kuhn-Tucker conditions presented in the preceding section. For inequality constraints, the LICQ conditions from definition 2.2.9 have to be satisfied which is accomplished by the active set method: Optimization progress can only be observed at the inequalities' limits, such that the whole problem is reformulated with equality conditions only and can then be treated as a simplified problem. Thus, the case with inequalities behaves like an equality constrained quadratic program, we can eventually ignore the inequality constraints that do not fall into the active set $I_A(\mathbf{x}^*)$ [109].

To keep things plain, we describe the SQP methodology with equality constraints. This is only exemplary, the case with inequality conditions is more relevant but very difficult to demonstrate in compact form. SQP methods are introduced as an application of Newton's method to the KKT optimality conditions. We consider

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^m} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & c_i(\mathbf{x}) = 0 \quad i \in I, \end{aligned}$$

where $f : \mathbb{R}^m \mapsto \mathbb{R}$ and $I = 1, \ldots, n$. Section 2.2.1.2 already defined the Lagrangian

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) \;=\; f(\mathbf{x}) - \boldsymbol{\lambda}^T \mathbf{c}(\mathbf{x}).$$

with $\boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_n]^T$. For ease of notation, we also define the gradient of the constraint

$$\mathbf{A}(\mathbf{x})^T := (\nabla c_1(\mathbf{x}), \nabla c_2(\mathbf{x}), \ldots, \nabla c_n(\mathbf{x}))\,,$$

for $\mathbf{c}(\mathbf{x}) = (c_1(\mathbf{x}), c_2(\mathbf{x}), \ldots, c_n(\mathbf{x}))^T$. By considering the first order KKT conditions, we obtain

$$f(\mathbf{x}, \boldsymbol{\lambda}) = \begin{pmatrix} \nabla f(\mathbf{x}) - \mathbf{A}(\mathbf{x})^T \boldsymbol{\lambda} \\ \mathbf{c}(\mathbf{x}) \end{pmatrix} = \mathbf{0}. \tag{2.7}$$

The usual approach to solve such nonlinear systems is Newton's method. Consider the Jacobian of (2.7)

$$\nabla f(\mathbf{x}, \boldsymbol{\lambda}) = \begin{pmatrix} \mathbf{W}(\mathbf{x}, \boldsymbol{\lambda}) & -\mathbf{A}(\mathbf{x})^T \\ \mathbf{A}(\mathbf{x}) & \mathbf{0} \end{pmatrix},$$

where $\mathbf{W}$ denotes the Hessian of the Lagrangian,

$$\mathbf{W}(\mathbf{x}, \boldsymbol{\lambda}) := \nabla^2_{xx} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}).$$

The Newton step from iterate $(\mathbf{x}_k, \boldsymbol{\lambda}_k)$ is given by

$$\begin{pmatrix} \mathbf{x}_{k+1} \\ \boldsymbol{\lambda}_{k+1} \end{pmatrix} = \begin{pmatrix} \mathbf{x}_k \\ \boldsymbol{\lambda}_k \end{pmatrix} + \begin{pmatrix} \mathbf{p}_k \\ \mathbf{p}_{\boldsymbol{\lambda}} \end{pmatrix}$$

where $\mathbf{p}_k$ and $\mathbf{p}_{\boldsymbol{\lambda}}$ solve the KKT system

$$\begin{pmatrix} \mathbf{W}_k & -\mathbf{A}_k^T \\ \mathbf{A}_k & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{p}_k \\ \mathbf{p}_{\boldsymbol{\lambda}} \end{pmatrix} = \begin{pmatrix} -\nabla f_k + \mathbf{A}_k^T \boldsymbol{\lambda}_k \\ -\mathbf{c}_k \end{pmatrix} \tag{2.8}$$

where $\mathbf{c}_k$, ∇f_k, $\mathbf{A}_k$ and $\mathbf{W}_k$ are vectors and matrices, respectively. The subscript indicate the gradient function values, constraint and constraint gradient values at the point $\mathbf{x}_k$. Usually, the matrix $\mathbf{W}_k$ is an approximation to the Hessian. The KKT matrix is nonsingular if i) the LICQ conditions hold, ii) the current iteration position $(\mathbf{x}_k, \boldsymbol{\lambda}_k)$ is close to the optimum $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$ and, iii) the second order sufficient conditions are satisfied at the solution. The Newton iteration can then be shown to be quadratically convergent.

A particular difference between Newton's method and SQP methodology is that in the latter a quadratic approximation to the objective function f is employed. The task is then (cf. section 2.3.1.1)

$$\begin{aligned} \min_{\mathbf{p}} \quad & m_k(\mathbf{p}) \\ \text{subject to} \quad & \mathbf{A}_k \mathbf{p} + \mathbf{c}_k = 0. \end{aligned}$$

where

$$m_k(\mathbf{p}) = f_k + \nabla f_k^T \mathbf{p} + \frac{1}{2}\mathbf{p}^T \nabla^2 f_k \mathbf{p} \tag{2.9}$$

In each iteration step, a quadratic model has to be set up and is subject to minimization. Assuming solvability, it can be shown that this system has a unique solution $(\mathbf{p}_k, \boldsymbol{\mu}_k)$ that solves

$$\begin{aligned} \mathbf{W}_k \mathbf{p}_k + \nabla f_k - \mathbf{A}_k^T \boldsymbol{\mu}_k \\ \mathbf{A}_k \mathbf{p}_k + \mathbf{c}_k = 0. \end{aligned}$$

This is equivalent to (2.8): By subtracting $\mathbf{A}_k^T \boldsymbol{\lambda}_k$ from the first equation of both sides of (2.8) we have

$$\begin{bmatrix} \mathbf{W}_k & -\mathbf{A}_k^T \\ \mathbf{A}_k & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{p}_k \\ \boldsymbol{\lambda}_{k+1} \end{bmatrix} = \begin{pmatrix} -\nabla f_k \\ -\mathbf{c}_k \end{pmatrix},$$

thus identifying $\boldsymbol{\lambda}_{k+1} = \boldsymbol{\mu}_k$. Hence, the above indicates the equivalence of Newton's method and the SQP method. This is useful since there are many theoretical results for Newton's method available. In practice, quadratic models like (2.9) have to be solved in each iteration step.

2.3.1.4 Derivative-Free Optimization

Derivative-free optimization (DFO) belongs to the class of trust-region methods. It is designed for unconstrained problems which have long function evaluation time and no analytical derivatives available. Also, it is a practical realization of the optimization algorithms discussed in the previous sections. We will apply the DFO algorithm as to compare other algorithms proposed in the ongoing sections. For now, we briefly outline the methodology. The implementation is detailed in Conn et al. [21–23], Scheinberg [133] and based on methods from Lawrence et al. [93] and Armijo [6].

DFO Approximation The algorithm is based on a substitute model, which is a quadratic approximation with polynomial basis $\{\phi_i(.)\}_{i=1}^q$. Now, the approximation model is determined by fulfilling that the approximation model interpolates exactly at the interpolation points $\{\mathbf{x}_j\}_{j=1}^p \subset \mathbb{R}^m$. In other words, we seek a quadratic model $m(\mathbf{x})$ such that

$$m(\mathbf{x}_j) = f(\mathbf{x}_j) \qquad j = 1, \ldots, p.$$

with $f : \mathbb{R}^m \mapsto \mathbb{R}$. Since $\{\phi_i(.)\}_{i=1}^q$ is a basis of in the space of quadratic polynomials, we seek the coefficients α_i from

$$f(\mathbf{x}_j) = \sum_{i=1}^{q} \alpha_i \phi_i(\mathbf{x}_j) \qquad j = 1, \ldots, p.$$

The coefficients $\alpha = (\alpha_i \ldots, \alpha_q)$ in the approximation model are then obtained via

$$\boldsymbol{\Phi}\alpha = \mathbf{f}.$$

The determinant of the coefficient matrix

$$\mathbf{\Phi} = \begin{pmatrix} \phi_1(\mathbf{x}_1) & \cdots & \phi_q(\mathbf{x}_1) \\ \vdots & & \vdots \\ \phi_1(\mathbf{x}_p) & \cdots & \phi_q(\mathbf{x}_p) \end{pmatrix}$$

necessarily needs to be non-zero. This is fulfilled when the number of interpolation points and the cardinality of the basis are both equal. For a full quadratic interpolation, p equals $\frac{1}{2}(m+1)(m+2)$. In DFO, the basis are fundamental Newton polynomials which are produced by a Gram-Schmidt orthogonalization [93]. The coefficients α_i can now be used in the model:

$$m(\mathbf{x}) = \sum_{i=1}^{q} \alpha_i \phi_i(\mathbf{x}) \tag{2.10}$$

Trust Region Technique Trust regions techniques are in particular well suited for nonlinear problems. The idea of trust region means that the general quadratic approximation model

$$m(\mathbf{x}_k + \mathbf{s}) := f_k + \nabla f_k^T \mathbf{s} + \frac{1}{2}\mathbf{s}^T \nabla^2 f_k \mathbf{s}$$

is only assumed to be accurate for small $||\mathbf{s}||$. Thus, the approximation should only be used in a region where we can 'trust' the approximation. This trust region is a n-dimensional ball

$$B_{\Delta_k}(\mathbf{x}_k) := \{\mathbf{x} \in \mathbb{R}^n | \ ||\mathbf{x} - \mathbf{x}_k|| \leq \Delta_k\},$$

with Δ_k called the trust region radius. Here, we first choose a maximum distance - the trust region radius - and then seek a direction to attain the best improvement possible, subject to this distance. If this step proves to be unsatisfactory, we reduce the distance measure and try again. The algorithmic details are given in the implementation papers [21, 93].

DFO Minimization Assuming we have a model for the objective function f, the next task is to minimize the model. Since we can trust the approximation only near the current point, i.e. the midpoint $\mathbf{x}_c$ of the interpolation points, we reformulate the optimization problem as

$$\begin{array}{ll} \text{minimize} & m(\mathbf{x}) \\ \text{subject to} & ||\mathbf{x} - \mathbf{x}_c|| \leq \Delta_k \end{array}$$

where the trust region condition acts as a nonlinear inequality constraint. This inequality constraint requires the fulfillment of the Karush-Kuhn-Tucker conditions and thus the SQP algorithm finds application. More specifically, the implemented SQP algorithm is a variant ensuring the search moves in feasible directions (Feasible SQP - FSQP [6, 93]).

2.3.1.5 Evolutionary Algorithms

Evolutionary algorithms (EA), also known as direct search methods, essentially spread parameters in the space of potential solutions to a parametrized problem. EA take evolution in nature as a role model and so establish a heuristic and conceptional simple method to optimize any parametrized problem in a large number of disciplines. These algorithms are proposed for optimization problems where conventional methods like gradient search are not suitable or even not applicable at all. They formalize what is observed and known in

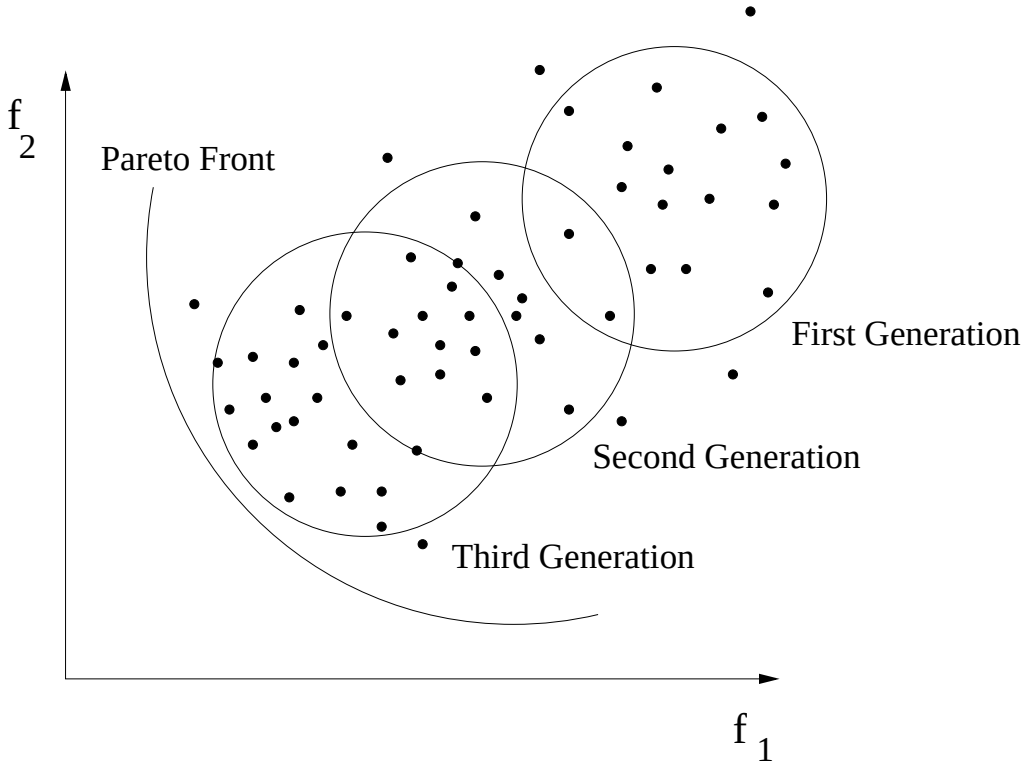


Figure 2.5: Evolutionary Algorithm with three generations moving towards the Pareto front

nature as mutation, recombination and selection. These principles define rules for varying parameters and the way these parameters develop towards an optimal state according to certain criteria. Figures (2.5) and (2.6) indicate the iterative EA process. The mutation and crossover operators are used to create new parameter combinations suggested as potential solutions to the optimization problem. Bäck [7] and Deb [30] overview EA's and thereby cover a range of EA applications.

EA split into genetical algorithms (GA) and evolutionary strategies (ES). Both GA and ES use mutation and recombination. The difference between genetical algorithms and evolutionary strategies stem from the numerical representation of the design variables, so ES works on the real valued string or vector of design variables while GA are encoded as bitstrings.

Figure (2.7) shows a one-bit mutation of a bitstring and figure (2.8) a one-point crossover between two bitstrings. Each bitstring corresponds to a unique real number and thus,

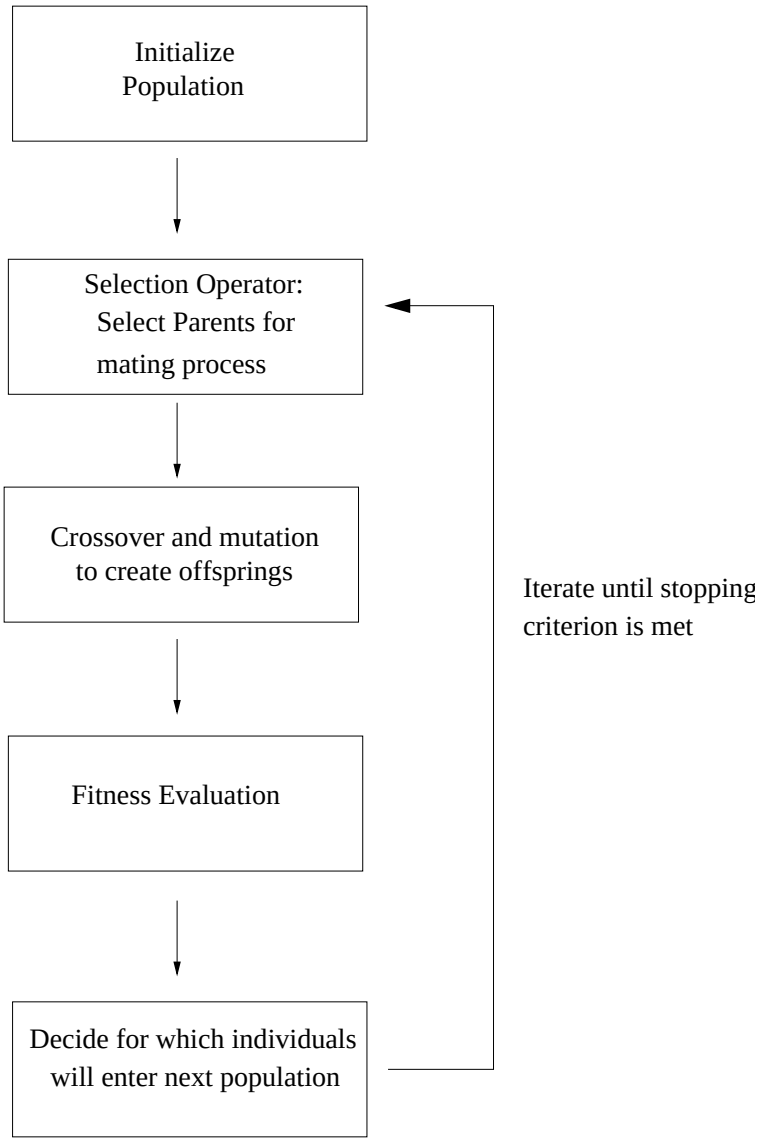


Figure 2.6: Flowchart Evolutionary Algorithm

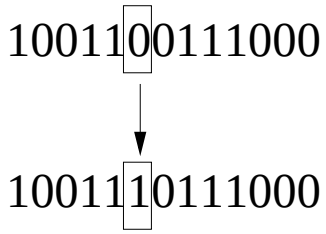


Figure 2.7: One-Bit Mutation of a Bitstring

the mutation and recombination produced new real values, the 'offsprings' of the original strings. In evolutionary strategies, mutation and recombination is performed, for instance, via Gaussian deflection and arithmetic mean. We will show an example of an evolutionary

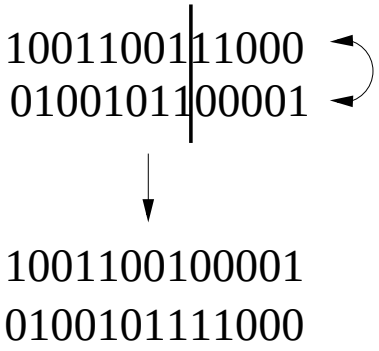


Figure 2.8: One-Point Crossover of Bitstrings

strategy later this section. However, in this work we will generally refer to evolutionary algorithms since GA and ES can be seen as slowly merging together.

EA differs from conventional techniques in several ways, most distinguishing they i) use a string or a vector to represent the decision variables, ii) operate on a set of solutions, thus exploring several points simultaneously, and iii) use operators that mimic some of the processes of natural evolution. The biological background is covered in Fogel and Corne's book [46]. We refrain from a further literature review since section 1.2 already presented a state-of-the-art review.

Evolutionary algorithms provide general heuristics for exploration and thus serve to exploit the design space in a randomized manner. The meaning of the mutation and recombination operators is to enlarge the diversity of the population, and hence the covered search space. Each individual in the population will then be evaluated and assigned a fitness. Based on this fitness, the selection operator decides which individuals are maintained and carried over to the proceeding generation, simulating natural evolution as an iterative and stochastic computing process. Although conceptually simple, these algorithms are sufficiently complex to provide robust and powerful search mechanisms. EA parameters are of important relevance, critically determining overall performance. They include population, offspring and generation size, recombination, variation and mutation operators with parameters therein, variable distribution, recombination, and mutation probabilities.

In ES, the minimal concept for imitating natural evolution is the (λ, μ) strategy, generating λ offsprings from μ parents and applying the selection operator to these λ offsprings. We here present the basic $(\lambda, \mu + \lambda)$-ES with $\mu = 1$ which is characterized through one parent in each generation producing λ offsprings. The set of the next iteration consists in the union of the parent and the $(1 + \lambda)$ offsprings. This set is then subjected to the selection operator. Figure (2.5) gives an impression of the process leading a set of individuals towards an optimal state, the Pareto front. The strategy is also referred to as two-member evolutionary strategy and was proposed by Rechenberg in 1973 [123] and recently applied for structural mechanics problems by Papadrakakis [114].

In order to categorize different evolutionary algorithms, we state the common methodology to which we will later refer when employing evolutionary models [7]:

$g = 1$;
initialize P_g
evaluate P_g
while { criterion } do
$P_{g,R} = R(P_g)$
$P_{g,M} = M(P_{g,R})$
$P_{g,V} = V(P_{g,M})$
$P_{g+1} = S(E(P_{g,V} \cup Z))$
$g \leftarrow g + 1$
end

where R, M ,V and S denote operators for **R**ecombination, **M**utation, additional **V**ariation, and **S**election, respectively. **E**valuation is realized in each iteration step and Z serves as further selectable set, e.g. $Z = P_g$ results in the $(\lambda, \mu + \lambda)$ strategy where λ individuals create μ offsprings and the selection is applied to $\mu + \lambda$ individuals. P_g denotes the whole population in generation g and $P_{g,\cdot}$ the subpopulation generated by the corresponding operators. The population consists of individuals

$$P_g = \{p_{g,1}, \ldots, p_{g,m}\},$$

correspondingly

$$P_{g,R} = \{p_{g,R,1}, \ldots, p_{g,R,m_R}\}$$

for all subsets. Each population element is a vector

$$p_{g,i} = ((p_{g,i})_1, \ldots, (p_{g,i})_n)^T,$$

respectively

$$p_{g,R,i} = ((p_{g,R,i})_1, \ldots, (p_{g,R,i})_n)^T,$$

for each subset. Here, m denotes the number of population members in the initial population, m_R is the population after recombination, and n is the length of the parameter vector.

The $(\lambda, \mu + \lambda)$ evolutionary strategy applies the identity for recombination and variation, i.e. $R = I$ and $V = I$. The population is initialized in each generation by one member only

$$P_g = \{p_{g,1}\}$$

which then produces a population by the mutation operator only. The evolutionary strategy we apply utilize the following operators (also confer [112, 113]):

1. Mutation: The parent (here: $p_{g,1} \in P_g$) of the generation g produces an offspring $p_{g,M,k} \in P_{g,M} \quad \forall k$ through

   $$p_{g,M,k} = p_{g,1} + \mathbf{z}$$

   where $\mathbf{z} = (z_1, z_2, \ldots, z_n)^T$ is an n-dimensional random deviation vector according to a specified distribution. This operation generates each element of the set $P_{g,M}$ and thus needs to be applied m_M times, giving a set of $m_M = \mu$ new elements.

2. Evaluation and Selection: The evaluation and selection operators are defined to pick the one best fit and viable element from the population, expressed as

$$S(E(P_{g,M} \cup Z)) = \{p_g^* \in E(P_{g,M} \cup Z) \mid f(p_g^*) = \min\{E(P_{g,M} \cup Z)\} \text{ and } p_g^* \text{ satisfies all constraints}\}$$

The appropriate choice for the deviation vector $\mathbf{z}$ in Step 1 is a uniform or Gaussian distribution obtained via a latin hypercube sampling algorithm. Obviously, the uniform distribution reaches a more extended distribution of parameters and has the advantage to cover the search space more thoroughly. The Gaussian distribution, on the other hand, scatters more closer around the initial parameter and can so serve to refine the search in the later stages.

Simulated Annealing We also utilize a simulated annealing (SA) approach, developed by Kirkpatrick in the 1980's [86]. This is an evolutionary strategy where the standard deviation adapts in each generation according to a certain criteria. The simulated annealing approach may also be compared to the $(1, \mu + 1)$ strategy described above. We decided to choose the standard deviation σ in each generation as

$$\sigma_{i+1} = \frac{\sigma_i}{j} \quad \text{for} \quad j = 1, \ldots, n$$

with n indicating the number of generations. A stronger reduction in the standard deviation can be achieved by

$$\sigma_{i+1} = \frac{\sigma_i}{2^j} \quad \text{for} \quad j = 1, \ldots, n.$$

The recombination and mutation operators can be applied for the SA algorithm as described before.

2.3.2 Methods for Optimizing Multiple Objectives

Usually, multiple objectives are accumulated into one objective through a linear combination and are then optimized by a standard single-optimization routine. This roughly describes the classic approach when multiple objectives appear, although, the difference from optimizing single to multiple objectives is vital in the optimization process. We briefly address a classic multi-objective approach and indicate characteristics and involved drawbacks by reviewing Deb [31], Horn [70], and Schwehm [139]. The cited literature also gives a more detailed overview and discussion of several of these deterministic methods on multi-objective optimization. The best known and usually applied method to render a multi-objective optimization problem is the weighted sum method, which cumulates all objectives $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_n(\mathbf{x}))$ into one by a linear combination via the normalized weight vector $\mathbf{w} = (w_1, \ldots, w_n)$, thereby defining a new function

$$F(\mathbf{w}, \mathbf{f}) := w_1 f_1 + \cdots + w_n f_n.$$

F is then optimized by any single-objective optimization algorithm given, for instance, in Nocedal and Wright [109].

Each weight vector $\mathbf{w}$ corresponds to exact one point along the Pareto front. It can be shown that, *if* $F(\mathbf{w}, \mathbf{f})$ converges, it converges to a Pareto optimal point. Now, by varying the vector $\mathbf{w}$ and utilizing a single-objective optimization algorithm applied to each proposed design $\mathbf{x} \in \mathbb{R}^m$, we may end up with a set of solutions which are all optimal in the Pareto sense. This approach is conceptionally simple, but causes several drawbacks,

1. Since the weights reflect a set of fixed priorities, the preferences of the decision-maker is unintentionally taken into account.

2. There is no clear relationship between the weightings and the corresponding solution.

3. In lack of this coherence, the runs are usually performed independently from each other, thus synergies can usually not be exploited which, in turn, may cause high computational overhead.

4. How to determine the weightings from the decision-makers' preferences is an ad-hoc procedure.

5. It is not possible to locate solutions at non-convex parts of the Pareto-front.

6. If the decision maker changes preferences after the optimization procedure is finished, the Pareto search needs to be performed again. This is especially problematic when function evaluations and thus the overall computing process are time-consuming as it is the case in engineering applications.

Das and Denis [24] and Steuer and Choo [145] review drawbacks of this method.

The weighted sum approach can easily be extended to higher-order methods by minimizing L_p metrices,

$$\left(\sum_{i=1}^{n} w_i |f_i - z_i|^p \right)^{\frac{1}{p}},$$

where the optimal value z_i for each function f_i has to be known. Deb [31] also discusses several variants from this class. Another technique is the ϵ-constrained approach where all but one objectives are formulated as constraints to the optimization problem. Again, this defines a single-objective optimization problem with additional constraints. The problem can then be read as follows:

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^m} \quad & f_j(\mathbf{x}) \\ \text{subject to} \quad & f_k(\mathbf{x}) \leq \varepsilon_k \quad k = 1, \ldots, n \ \text{ and } \ k \neq j \\ & c_i(\mathbf{x}) \geq 0, \quad i \in I_I \\ & c_i(\mathbf{x}) = 0, \quad i \in I_E \end{aligned}$$

By this conversion, the feasible space is narrowed since it is necessary to fulfill a number of artificial constraints. The task of the problem is now to find the solution which minimizes this feasible region.

In comparison to the weighted sum approach, the ϵ-constrained method is able to approximate non-convex regions of the objective space. On the other hand, as the weighted sum technique requests to determine weights, the user is demanded to provide the ϵ-bounds. Both are interventions and introduce subjective information which may hinder the optimization process to succeed. We continue the discussion in section 2.6 of this chapter.

2.3.3 Approximation Models

Approximation models set up a surrogate to the actual objective function. The modeling procedure aims to require only very few costly function evaluations such that the overall optimization process can save computing time. After setting up, the model is optimized involving an arbitrary number of computationally cheap function evaluations. Moreover, the sampling points can be produced in a randomly or deterministic manner. The substitute models we employ here are combined with evolutionary algorithms to obtain a global and efficient search procedure.

The approximation model used in the response surface method is widely known and most often applied in the context of reliability analysis. The objective of this method is to obtain a description of the influence of each variable and their possible combinations on the response of the system, achieved via a polynomial interpolation of input/output sample points. This regression then serves as a substitute for function evaluations. The response surface method is described in Shyy et al. [140] in context of aerodynamics applications. Furthermore, kriging models [122] are specialized response surface approaches and consist of a global and a local model where the local model is updated with each new iteration.

Another approach is to use a reduced model. This reduced model can be a less precise evaluation of the individuals, such as a numerical evaluation working on only very few unknowns. After deciding in which part of the fitness landscape a more exact evaluation is needed, one can apply the full, or exact model in these regions. Giannakoglou [85] suggest to use inexact pre-evaluation in which the evolution is based on exact and inexact evaluations and the approximate model is working as a pre-evaluator with tentative steps.

The approximation modeling approach can be compared to the quadratic interpolation from the SQP method. In contrast, the approximation model is here an orders of magnitudes more demanding and complex process. Whereas the quadratic model in the trust region approach is obtained by three function evaluations, approximation with hundreds of sampling points (in higher dimension) via polynomials is not practical anymore.

Inspired by these modeling techniques, it is self-evident that introducing an efficient function evaluation substitute other than of polynomial quality may yield an improvement concerning the approximation capability: Neural networks are capable to map the complex

functional coherence between the model parameters and the objective function requiring less computation effort than a polynomial approximation. In general, neural networks provide a rapid mapping of a given input into the desired output quantities. Admitting that the predicted results fall within acceptable tolerances, this method can produce simulation output in orders of magnitude less computational effort than the conventional computing process [113]. Most often radial basis function networks are used, they allow a finer resolution of the function surface in clusters in the output space.

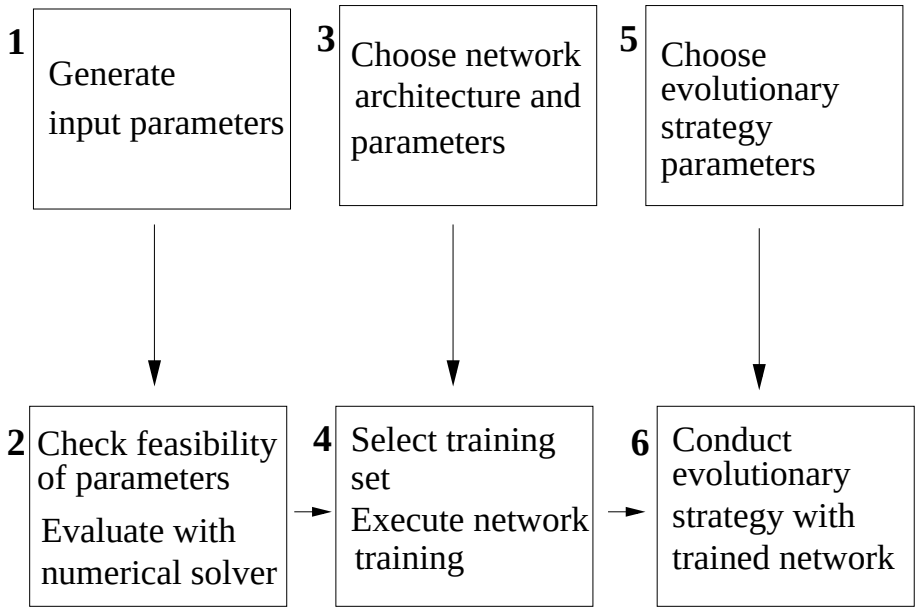


Figure 2.9: Network Training. The upper three operations are independent from each other.

A neural network has to be 'fed' with function evaluations from a numerical solver and is then trained to adapt to the functional coherence. The selection of training data is crucial in this process where we investigate different alternatives: The parameters may be chosen by an evolutionary algorithm or by a certain application of efficient Monte Carlo generated parameter samples. Figure (2.9) indicates the process when a network approximation model is used. After the network is trained, the evolutionary strategy is applied as shown in figure (2.10).

Giannakoglou [52] reviews the most common approximation models and gives a literature overview with applications in engineering context. Papadrakakis et al. [113, 114] as well as Hurtado et al. [76, 77, 111] and other authors [16, 52, 107, 110] applied a conjoint evolutionary strategy - neural network algorithm to engineering relevant applications. A more recent account can be found in Duvigneau and Visonneau [37] and Giannakoglou [85]. Applications are often from structural mechanics, employing a linear elastic material model. The present work aims at optimizing flow process which have a considerable more demanding underlying behavior. In chapter 4, we apply Bayesian regularization networks and adaptive neuro-fuzzy inference systems as approximation models. Due to advanced generalization features the mentioned networks possess more intelligent approximation capabilities than the plain networks.

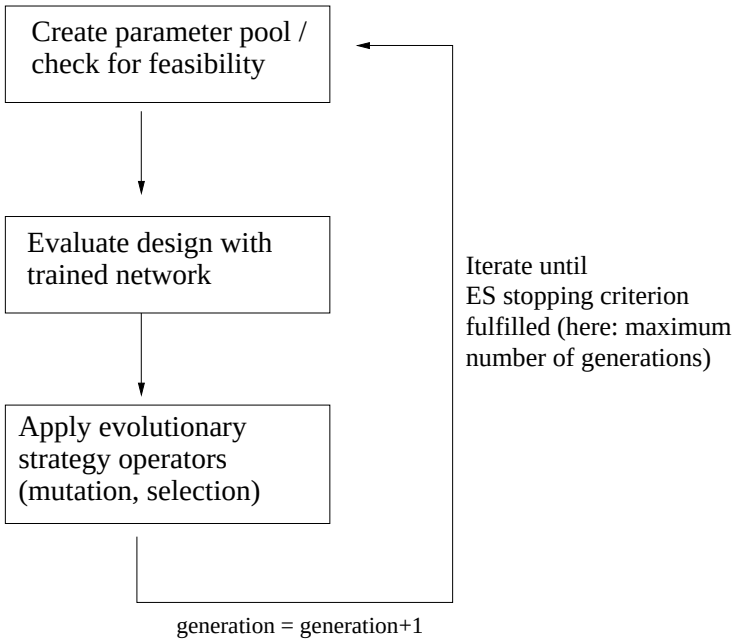


Figure 2.10: After training the model is used in an evolutionary strategy.

Latin Hypercube Sampling

All data samples are generated by a Monte-Carlo simulation that employs a latin hypercube sampling (LHS) procedure. LHS means that each subset number of each random variable is combined with other subset numbers of the rest of the variables only once in a randomized way. This approach not only guarantees the exhaustive exploration of the design space, it also maximizes the marginal distribution of the given random variables and thus allows for a reduction of the sampled population. We emphasize that efficient sampling is an important aspect in the optimization procedure since it provides the basis for successful network training. We decided for two ways of selecting the training set (see also [76, 114]), which are described in the following.

Evolutionary Network Formation

By combining approximation models with evolutionary algorithms, there appear several strategies for training the model. In order to obtain a finer resolution near the global minimum, we aim to generate more sampling points near this minimum. This is achieved by choosing the training set by using data from evolutionary strategy optimization steps until the computed designs arrive at a region near the optimum. We denote this method by evolutionary network formation (ENF). The ENF training set selection is accomplished by the following algorithm 1, where the term 'model' refers to a specific neural network, and ES stands for evolutionary strategy.

Algorithm 1: Evolutionary Network Formation

1. ES optimization phase:
   - 1.1. choose parameters for ES, here: λ, μ, mutation operator M.
   - 1.2. execute ES
2. Model training phase:
   - 2.1. choose model architecture and corresponding parameters
   - 2.2. choose sampling data from ES phase
   - 2.3. execute training
   - 2.4. test model by a validation set, if tolerance is not met reject and goto step 2.2.
3. ES optimization with trained model:
   - 3.1. choose (new) ES parameters
   - 3.2. use model as solver substitute conducting ES

Network Feeding Training

Here, the training set is chosen based on a Gaussian or uniform latin hypercube sampling distribution of the design variables around the midpoints (zero-vector of the parameterized design) of the design space. This method is briefly denoted by Network Feeding (NF). The procedure is described in algorithm 2.

2.4 Evaluation

In the context of the present work, the term evaluation refers to how to evaluate the objective function. In shape design problems, the objective is a function of the solution of a complex differential equation system. This solution is obtained via a numerical approximation which is quite computationally expensive and a large scientific field by its own means.

We thus first define the governing equations from fluid mechanics and mention the employed numerical methods. Thereafter, the structural mechanics case is considered.

2.4.1 Fluid Mechanics Problems

The flow solver is based on the numerical solution of the balance equations for mass, momentum, and energy for an incompressible Newtonian fluid. Assuming a steady flow

Algorithm 2: Network Feeding

1. Data set selection:
   - 1.1. choose input parameter set via Monte-Carlo simulation
   - 1.2. evaluate parameter vectors
2. Model training phase:
   - 2.1. choose model architecture and corresponding parameters
   - 2.2. choose training and validation data set
   - 2.3. execute model training
   - 2.4. test model by a test set, reject and goto step 2.2. if tolerance is not met
3. ES optimization with trained model:
   - 3.1. choose ES parameters
   - 3.2. use model as solver substitute conducting ES

and neglecting buoyancy effects the equations take the following form:

$$\begin{aligned} \frac{\partial v_i}{\partial x_i} &= 0\,, \\ \frac{\partial\left(\rho v_j v_i\right)}{\partial x_j} - \frac{\partial}{\partial x_j}\left[\mu\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\right)\right] + \frac{\partial p}{\partial x_i} &= 0\,, \\ \frac{\partial(\rho c_p v_i T)}{\partial x_i} - \frac{\partial}{\partial x_i}\left(\kappa \frac{\partial T}{\partial x_i}\right) &= 0\,, \end{aligned} \tag{2.11}$$

where v_i are the velocity vector components with respect to the Cartesian coordinates x_i, p is the pressure, T is the temperature, μ is the dynamic viscosity, ρ is the density, κ is the thermal conductivity, and c_p is the specific heat at constant pressure. The index corresponds to $i = 1, 2, 3$ indicating operations in three dimensions, and the Einstein summation convention is used. For simplicity, we assume that all material parameters are constant.

Numerical Treatment Within the optimization procedure a relatively large number of individual flow simulations may be necessary and the shape variations may lead to "nasty" grids. Thus, an important prerequisite for the flow solver is its numerical efficiency and robustness, in particular, also in situations when the grid is distorted.

The flow evaluations in this work employ the finite-volume solver FASTEST 2-D [78] which specially takes into account these aspects. It is based on a fully conservative second-order finite-volume scheme for general non-orthogonal block-structured grids (see Ferziger and

Perić [45]). A special interpolation is employed for approximating cell face values by nodal values within the finite-volume discretization proposed in Lehnhäuser and Schäfer [94]. This is based on a multi-dimensional Taylor series expansion and ensures second-order spatial accuracy regardless on the grid distortion.

The pressure-velocity coupling is established by using a special variant of the SIMPLE algorithm (see Patankar and Spalding [116], Lehnhäuser and Schäfer [95]). To solve the various sparse linear systems within the pressure-correction scheme an iterative ILU decomposition following Stone [147] is used. For convergence acceleration a nonlinear multi-grid scheme is implemented [73], which has proven its high numerical efficiency for many applications (e.g. [35, 74, 129, 131]).

2.4.2 Structural Mechanics Problems

We briefly indicate the equations from linear elasticity theory which will be applied for the structural mechanics computations. For linear elastic material, we have for the strain-displacement relation

$$\varepsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right).$$

Hooke's law gives a connection between stress ε_{ij} and strain σ_{ij}, which is described by

$$\sigma_{ij} = \lambda \varepsilon_{kk} \delta_{ij} + 2\mu \varepsilon_{ij}$$

where λ and μ are the Lamé-constants

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \quad \text{and} \quad \mu = \frac{E}{2(1+\nu)},$$

with E the elastic modulus and ν the Poisson ratio. The equilibrium can then be derived to

$$\frac{\partial \sigma_{ij}}{\partial x_j} + \rho f_i = 0,$$

inserting yields

$$(\lambda + \mu)\frac{\partial u_j^2}{\partial x_i \partial x_j} + \mu \frac{\partial^2 u_i}{\partial x_j \partial x_j} + \rho f_i = 0. \tag{2.12}$$

The boundary conditions are Neumann or Cauchy conditions, describing either the exterior applications of forces or movements on the boundary.

Numerical Treatment Computations for the structural mechanics examples are produced with ANSYS© [3] which has a qualified grid generator as well as efficient numerical routines implemented to tackle linear elastic 2-D computations.

2.5 Development

This section investigates how the development of new designs is accomplished. The numerical evaluation requires a structured grid for efficiency reasons. Considering that, it is not obvious how to deform the grid without violating the numerical solution too much. We used the free-form deformation technique which has the advantage that most desirable grid features are maintained. Figure (2.12) gives an example on how a given geometry is deformed via this approach.

Geometry Representation and Variation

The shape representation method and so the search space of the algorithm necessarily needs to express accurately enough the global optimal shape, otherwise the algorithm would certainly fail to find the optimum shape design. On the other hand, the search space should be as small as possible. This limits the number of possible shape designs and will lead the optimization algorithm faster to the desired optimal shape. The correct shape representation method allows us to introduce our problem specific knowledge to the search process. This means to limit the search space without limiting the accuracy of the optimization result. Moreover, the representation method should prevent the creation of invalid shapes which not only waste computation time, but also might lead to an optimal but invalid shape. The parameterization of the geometry is obtained via splines which make it possible to express the connecting shape in parametric form (Samareh [127]). The deformation of the shapes is described by Bézier-Splines: The characteristics of a Bézier curve is that the polygon spans a convex hull in which the curve is contained. Furthermore, if one control point is moved, the whole curve is altered. The presented methodology is also known as the free form deformation (FFD) technique as described in Harzheim et al. [63]. According to the cited authors, we define the vector

$$\mathbf{G} := (\mathbf{x}_1, \ldots, \mathbf{x}_n)^T,$$

containing the coordinates $\mathbf{x}_i = (x_{i1}, x_{i2})^T,\ i = 1, \ldots, n$ of all grid points. We define shape basis vectors $\mathbf{S}_i$ displacing the grid coordinates from $\mathbf{G}$ to $\widetilde{\mathbf{G}}$ via

$$\widetilde{\mathbf{G}} = \mathbf{G} + \sum_{i=1}^{N} \alpha_i \mathbf{S}_i,$$

where N is the number of shape basis vectors and α_i are the corresponding deflection parameters (or, equivalently, design variables). In this context, design variables are parameters that the designer might 'adjust' in order to modify the system design. The shape basis vector $\mathbf{S}_i$ gives the direction and magnitude of the displacement at the ith control point. These control points are defined as

$$\mathbf{p}_{i,j} = \left(\frac{i}{I}, \frac{j}{J}\right), \quad i = 0, \ldots, I, \quad j = 0, \ldots, J, \text{ and } I + J = N,$$

indicating the equally spaced initial position. The condition $I + J = N$ means that N design points are portioned into each of their Cartesian directions. The deformation is then specified by moving the control points from their initial position $\mathbf{p}_{ij}$ to

$$\widetilde{\mathbf{p}}_{ij} := \mathbf{p}_{ij} + \Delta\mathbf{p}_{ij},$$

including a vector $\Delta\mathbf{p}_{ij}$ with the origin at the control point and pointing to the desired direction in which the mesh has to be displaced.

For simplicity, we consider a unit square and the canonical basis vectors in $\mathbb{R}^2$ e_1, e_2, such that every (grid-)point inside the square can be described by the linear combination $\mathbf{x}_i = s_i e_1 + t_i e_2$, $s_i, t_i \in [0, 1]$.

The assembly of the shape basis vector

$$\mathbf{S} = \left(\tau(\mathbf{x}_1), \ldots, \tau(\mathbf{x}_n)^T\right)$$

is then obtained from each shifted grid point,

$$\tau(\mathbf{x}_l) := \sum_{i=0}^{I} \sum_{j=0}^{J} B_{ij}(\mathbf{x}_l)\ \widetilde{\mathbf{p}}_{ij}. \tag{2.13}$$

The $B_{ij}(\mathbf{x})$ are Bernstein polynomials and are given by

$$\begin{aligned} B_{ij}(\mathbf{x}_l(s,t)) &= a_i(s)b_j(t) \qquad \text{where} \\ a_i(s) &= \binom{I}{i}(1-s)^{I-i}s^i \\ b_j(t) &= \binom{J}{j}(1-t)^{J-j}t^j \qquad \text{and} \\ \binom{k}{l} &= \frac{k!}{l!(k-l!)}. \end{aligned}$$

The above operations are only valid in a square. By a linear transformation one can transform any quadrangle onto a square, perform the shape deformation and transform back into physical space. In fact, this transformation to the standard square and back into physical space can be performed in one step, only taking effect on $\tilde{p}$ in (2.13).

By this technique, we convert a shape optimization task to a parameter value optimization task. We emphasize that the parametrization is crucial for finding the optimum. Figures (2.11) and (2.12) indicate a channel junction on which six deflection points are realized, three of them on each side of the connecting part of the channel junction.

The choice of parametrization certainly influences the optimal shape suggested by the algorithm. By any optimal shape we obtain, we always need to keep in mind that this is optimal in the corresponding design space. Changing the design space often gives different optimal shapes, so the challenging task is not to limit the possible results too much by the choice of parametrization. For a more detailed discussion about parametrization techniques we refer to Samareh [127].

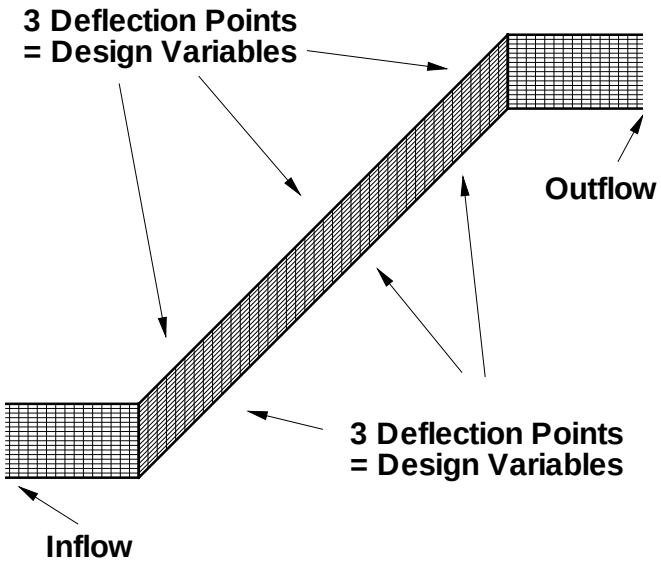


Figure 2.11: Staggered Channel Junction with six deflection points

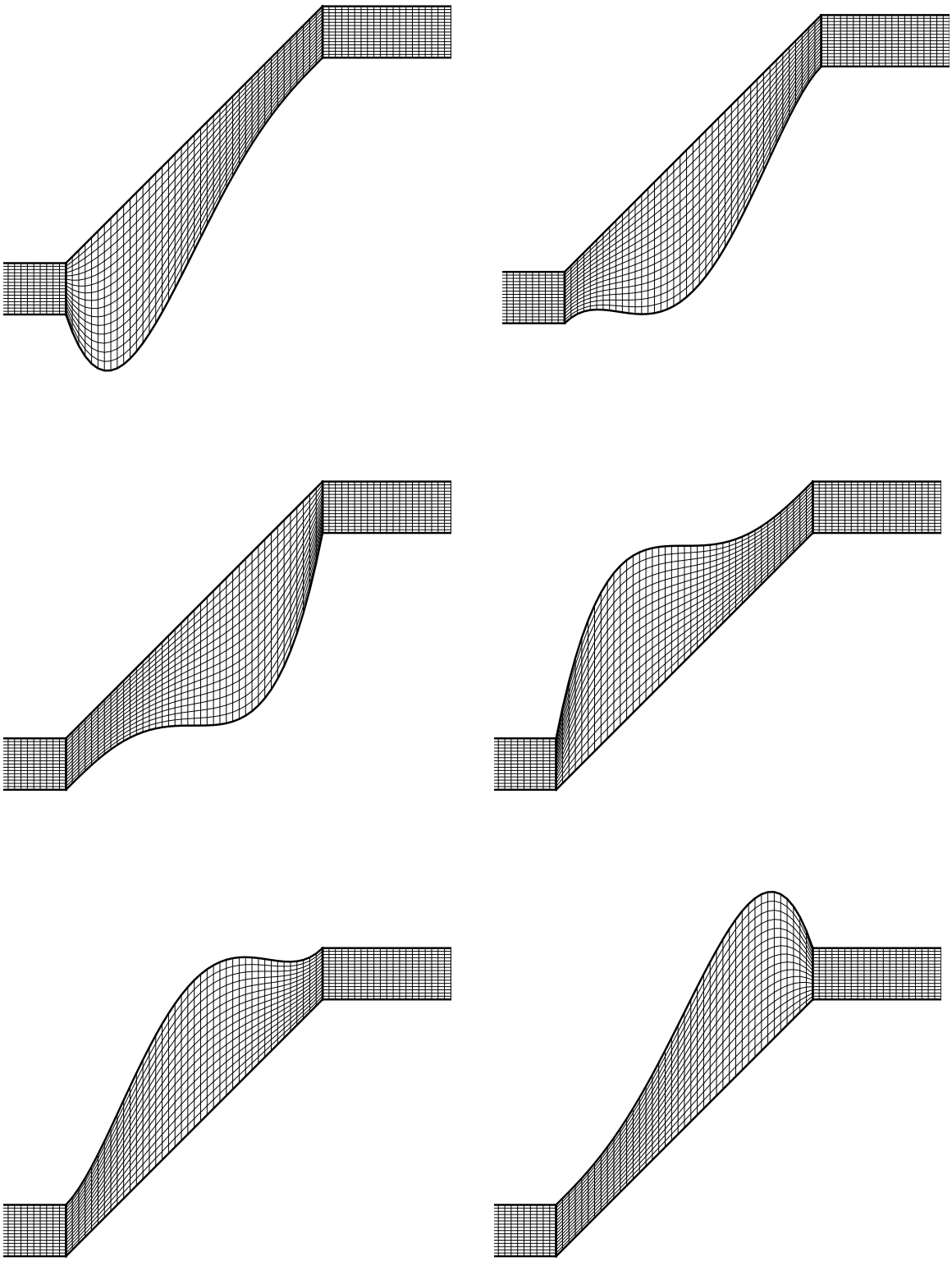


Figure 2.12: Effect of design parameters on shape variation (case with 6 deflection points).

2.6 Strategy: Discussion and Evaluation

In the former sections, we provided an insight into the components of which a typical engineering optimization process consists. In shape optimization, different design configurations cause different properties of the optimization model and the objective function. Also, the overall aim of the optimization process has a strong influence on the optimization model. As an example, consider optimizing an airfoil by defining the drag coefficient as a penalty to the lift coefficient. In this case, one changes an actual multi-objective problem into a single-objective problem, thus the obtained optimal solution must be seen as from an artificial optimization model not correctly depicting physical behavior.

We emphasize that these aspects have to be considered *before* the optimization process is executed. The crucial task is then to decide which optimization strategy is suitable for which kind of optimization model, resp. optimization problem. Again, most optimization problems have a multi-objective nature [105]. When only optimizing a single objective, one has to keep in mind that the optimization result always corresponds to the same artificial setup in which the optimization problem was stated in. Even though optimizing a single objective can be useful, the information provided by this optimum has to be augmented very carefully. Constraints, in this context, can have strong influence on the function characteristics and thus on the found global minimum. As we already pointed to, we usually do not know any functional characteristics of the objective in advance. That is why an optimization algorithm should allow to tackle the most general optimization model in which a given problem can be formulated. An overview and discussion of the proposed methodology is given, e.g. in Horn [70] and Deb et al. [31].

Another issue is the long function evaluation time. It is necessary to support the optimization process with an approximation model. Since plain neural networks have their strengths in pattern recognition tasks, their function approximation capabilities have to be examined separately. This approach has not found real appreciation in the scientific community and this is because that still, a network function approximation model demands an unbearable number of evaluations. We suggest to use highly sophisticated and progressive network models in this context. Whereas our numerical examination found that progressive models give reasonable results, the plain networks completely failed in this aspect.

Why Evolutionary Algorithms?

This section is an in-depth discussion of pros and cons using either EA or conventional methods for tackling multi-objective problems in engineering shape design. The choice of optimization algorithm is highly dependent on the nature of the underlying objective function. It is thus necessary to first define the kind of optimization problem to be solved.

The most concise features of the problems we consider in this work are the following:

- A considerable long function evaluation time.

- The objective function behavior, especially in conjunction with multiple objectives, is not known in advance.
- As we usually deal with parameterized problems, gradients are difficult and cumbersome to produce. A method working without derivative information is always considered more suitable to the problem at hand.
- It is difficult to learn about properties of design alternatives, for instance, a shape that reflects local optima and qualities of the design in aspects which are not subject to optimization, e.g. stability characteristics, is highly appreciated. Haase [55] presented an example where a local optimization result was preferred to a global one.

The classic approach to render an optimization problem having multiple objectives is to transform it into a single objective as the weighted sum approach and similar methods suggest. In order to obtain a search procedure, these local algorithms are started from several points distributed over the whole optimization region. This procedure is named "multistart", a very simple and intuitive global procedure. There is a number of drawbacks involved with this approach. We already mentioned some characteristics in section 2.3.2, and summarize them as follows.

- The outcome of such an optimization strategy obviously depends on the chosen weights or ϵ-value. The result is then strongly dependent on how the objectives are aggregated.
- If the Pareto front is non-convex, there might be no weight vector combination representing this region, thus not all solutions may be found.
- In a nonlinear optimization problem, different initial solutions do not guarantee finding different optimum solutions. This is particularly the case if the chosen inital solutions all lie in the basin of attraction of an identical optimum.
- This also means that a uniformly distributed set of weight vectors need not find a uniformly distributed set of Pareto optimal solutions.
- As in a multistart procedure, the algorithm has to be restarted from random initial solutions several times to hopefully find multiple optimal solutions. When many starting points are used, the same minimum will eventually be determined several times hence wasting computation time.
- Convergence to an optimal solution always depends on the chosen initial solution.
- Parallel processing when using deterministic transition rules is difficult, if not impossible to use.
- Furthermore, restarting the search process from different initial solutions in the decision space already constitutes a heuristic search.

In engineering practice, evolutionary algorithms can be used for parametrized optimization problems. Each parameter vector then represents, for instance, a flow geometry which can be evaluated by a numerical solver. The numerical evaluation provides a measure of design quality, e.g. in terms of performance of a heat exchanger. This performance then serves as the 'fitness' of the corresponding design object. We summarize reasons which make it useful, sometimes even necessary to employ heuristic strategies to the mentioned optimization problems.

- Especially in fluid flow shape optimization problems, where large gradients arise in the objective function, the derivatives are not known and have to be approximated requiring extra computational effort. It can be very computationally expensive to obtain these gradient information. EA do not require any gradient information and, therefore, may be efficiently applied to large-gradient or fluid flow problems.

- For multi-modal and multi-objective optimization strategies gradient based methods may quickly fail to provide satisfactory results. This is obvious since gradient based methods are considered as NP-hard in terms of computational complexity, i.e. the computation time for finding the global minimum increases exponentially with the dimension of the problem [7, pp. 52]. For higher-dimensional objective spaces, this may account for a huge overhead until convergence is reached by conventional optimization procedures. A gradient descent approach may be successful for the single-objective case, whereas in optimizing several objectives, the search procedure may be prohibitively expensive in terms of computation time (cf. [139, 156]). We cite Stützle [149] who thoroughly covered the computational complexity issue for algorithmic (in contrast to combinatorial) optimization problems.

- EA can be constructed to spread parameters in the search space and thus obtain global search properties. They are able to detect local as well as global optima.

- An important advantage is that EA are inherently designed for use on parallel machines. Especially the practicioneer, who has to deal with enormous computing times, strives to implement algorithms in parallel operation mode. The very fast progress in computer technology and decreasing hardware costs indicate a future trend towards parallel processing demands.

- The difference from single to multiple objectives is that the latter may not have a single solution which simultaneously satisfies all objectives to the same extent. A distinguishing feature of EA is that they work on a *set* of candidate solutions. This allows for the application in multi-objective optimization where a set of solutions is actually sought. An algorithm that emphasizes this problem aspect and giving a number of alternative solutions has a great practical value [161].

- In multi-modal problems, the objective function might have several local minima, which can simultaneously be detected by a single optimization run. Although one might most often be interested in the global optimum, knowledge of local minima can be very interesting since in engineering practice, a local minimum might have

e.g. stability properties that are desired in any context of the application. Knowing local minima corresponds to a thorough understanding of the design behavior.

- The no free lunch theorem by Wolpert and Macready [108, 158] says that, on average, all optimization algorithms have the same performance over all optimization problems. At least theoretically, it means that no 'optimal optimization' algorithm exists. It also implies that the most appropriate optimization algorithm depends on the nature of the problem and also the intention of the user: While algorithms based on gradient descent are most suitable for convex and unimodal problems, this is not anymore the case for multi-objective problems. In practice, it is clear that single-objective algorithms are no substitute for the evolutionary approach in optimizing multiple objectives. A single-objective algorithm cannot provide the same problem insight as a multi-objective algorithm does.
- EA are black-box methods. This means they are less problem-dependent than their gradient counterparts. They are also known as robust and easy-to implement methods which do not fail by applying them on different problems from the same class.

However, evolutionary optimization techniques have not yet found widespread acceptance in engineering design. This stems from the facts that in the single-objective case evolutionary algorithms involve more (expensive) function evaluations of the objective and, they are less accurate than gradient based methods. As a heuristic method EA are also criticized for having only a thin theoretical foundation. Furthermore, there are no optimal parameter settings known in advance (e.g. population size, selection of variation operators etc.).

2.7 Evolutionary Strategies in Application

This section states results from the application of evolutionary algorithms to shape optimization problems in structural mechanics. The presented methodology gives a first insight in applied optimization problems. We show some of the proposed methods from the preceding discussion. Structure mechanics serves as a first test of the proposed evolutionary strategies methodology since function evaluations are faster evaluated by the numerical solver than in fluid dynamics. We applied evolutionary strategies and simulated annealing where we designed the algorithms to use adaptive features for efficient EA parameter handling. In a multi-objective test case, we also used the weighted-sum approach which will be shown to give a good view on the Pareto alternatives. As we shall see, using the weighted-sum method will lead us to an extensive usage of function evaluations which are in this case not harmful since here, each function evaluation in structure mechanics is processed in a few seconds. The computations in this section are detailed in [134, 135].

We decided for the ANSYS structural mechanics software known as a reliable and fast finite element (FE) solver. To elucidate our results, we utilize MATLAB which offers easy-to-generate graphical features. ANSYS is employed for parametric modeling, meshing, and

FE simulation. The ANSYS version 5.7 (2000) is used, providing a probabilistic design toolbox which allows for parameter deflections according to some statistical distributions (e.g. Gaussian- or uniform distribution). The functional coherence exhibited by the FE simulations allows to apply a basic $(1, \mu + 1)$-ES. We here only realize the mutation and selection operators, neglecting recombination due to a large variety of parameter deflections which representatively covered the design space.

To generate random samples, we use latin hypercube sampling (cf. section 2.3.3), which allows for a further reduction of the sampled population, because each subset number of each random variable is combined with other subset numbers of the rest of the variables only once in a randomized way. This method maximizes the marginal distribution of the given random variables. The FE analysis is then performed with the corresponding set of points.

In our approach, model geometry coordinates are defined as control vertices to interpolation functions (e.g. B-Splines, NUBS, NURBS). This approach ensures to achieve a high geometric flexibility with a maximized boundary smoothness of the shape model. Among other advantages of these interpolation techniques, we emphasize the possibility to vary control points during the simulation without disturbing the consistency of the model [2]. ANSYS provides a powerful mesh generator used to remesh the whole geometry for each FE simulation.

2.7.1 Optimization of a Tensile Bar

The exemplary problem we decided for is a tensile bar with a centrical hole and a typical load parallel to the longitudinal axis (cf. figure (2.13)). Design variables are the inner vertices of the laterally bounding splines. These nine master coordinates have one degree of freedom, allowing deflection in x-direction while symmetry to the y-axis is enforced. The maximum occurring stress and structure volume are defined to be objective functions. According to equation (2.1), the optimization task reads:

$$\min_{\substack{x_l \le x_i \le x_u \\ i=1,\ldots,18}} \quad \sigma_{\mathrm{eqv,max}}, V,$$

with V denoting the interior area of the tensile bar. The constraints on the design variables are put into the decision space such that we have now to handle an unconstrained optimization problem. The constraints ensure that only reasonable designs are suggested to the numerical solver. Figure (2.14) shows five optimization runs started from different initial designs also indicated in figure (2.14). The $\mathrm{p}^{\mathrm{a}}_{1,1}, \ldots, \mathrm{p}^{\mathrm{e}}_{1,1}$ are initial points representing five different initial geometries. The final designs are indexed by $\mathrm{g_s}$ indicating that $\mathrm{g_s}$ generations were realized. A large value for the initial standard deviation is chosen.

2.7.1.1 Influencing Simulation Parameters

In the above algorithm different parameters which influence the optimization progress need to be discussed. Figure (2.13) displays the initial geometry and also indicates the

vertices which are subject to random deviations. In figure (2.14) we picture some possible geometries, each of which is represented by a vector containing the parameters defining a shape model. Surely, by placing the initial geometry close to the presumed optimal shape, we may gain an advantage concerning the number of simulation trials. We experimented different initial configurations to asses the robustness of the algorithm.

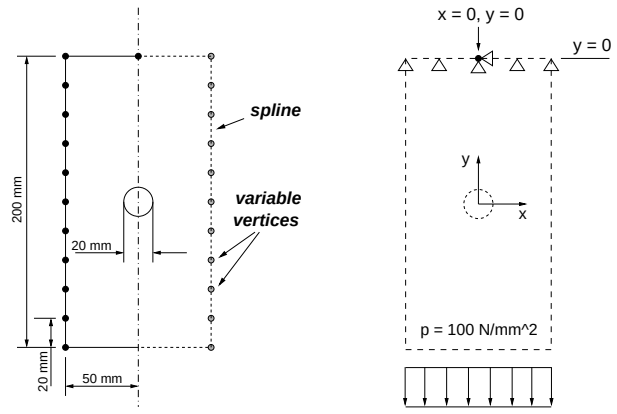
The statistical distributions which are applied on the design variables have a significant influence on the optimization progress and performance. Thus, a high standard deviation at the beginning of the iteration process is necessary to thoroughly explore the design space. In the subsequent generations the population is supposed to move closer to the desired optimum and then requires a low standard deviation to eventually 'hit' the desired optimal configuration. We applied the simulated annealing approach where we decreased the standard deviation in ongoing generations according to a specified rate.

We choose a population size of N = 30 in 14 generations to achieve a sufficient diversity in the design space. In the later runs, we might need to increase the population size (up to 50 individuals) since we like to elaborate a crisp Pareto front. The stopping criterion is the maximum number of generations but also very low iteration progress indicating convergence. This iteration progress is measured in terms of how far the best fit individual is distant from the other best fit individuals from the preceding clouds. The objective function values are the maximum stress and volume of the shape design, both obviously giving a trade-off identified in the appearing Pareto front.

2.7.1.2 Properties of the Optimization Approach

Since the main drawback of evolutionary algorithms is a usually high number of function evaluations, we here aim to keep the number of evaluations as low as possible. Important properties of our algorithm are

- *Adaption of the standard deviation* to explore the design space in early iteration stages and to refine the search in the later stages.



Mechanical Properties:	
Young's modulus	$2.1 \cdot 10^{11} \frac{N}{m^2}$
Poisson ratio	0.3
Thickness	5 mm

Figure 2.13: Test example: Tensile bar with centrical hole

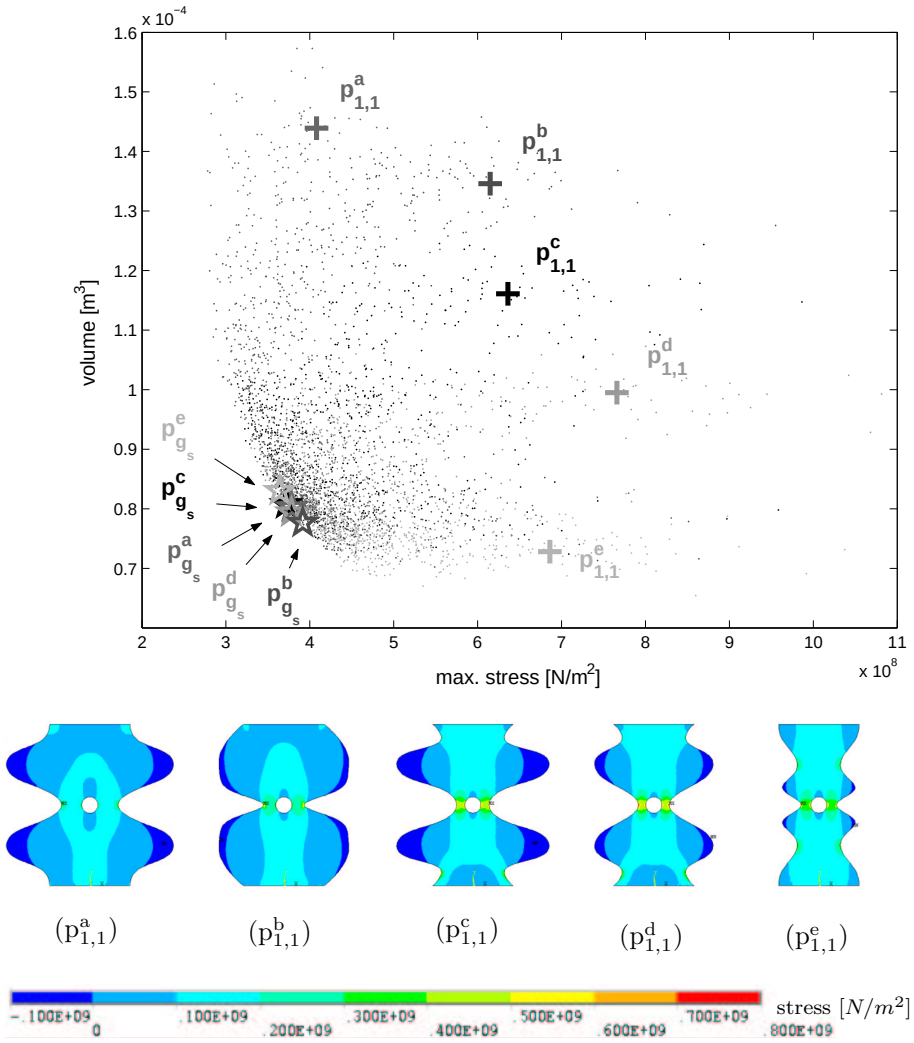


Figure 2.14: Optimization runs with different initial geometries

- *Adaption of the population size* gives us a tool to adapt and to exploit further optimization potential. It also helps to keep simulation runs at a minimum, if a further increase in the objectives is not possible.

- *Stopping criterion* given by the maximum number of trials (generations and individuals, respectively).

In each generation, the above parameters are updated and adapted according to the optimization performance. If the optimization progress stalls in the late runs, the standard deviation should be lowered, since the designs need further refinement which cannot be provided by coarse deflections.

2.7.1.3 Multi-Objective Optimization

We schematically indicated an emerging Pareto front in figure (2.15) which can also be seen in our computations in figure (2.14). The parameter adaption successfully approached the alternatives along the physical design constraints. The properly ES parameter adaption

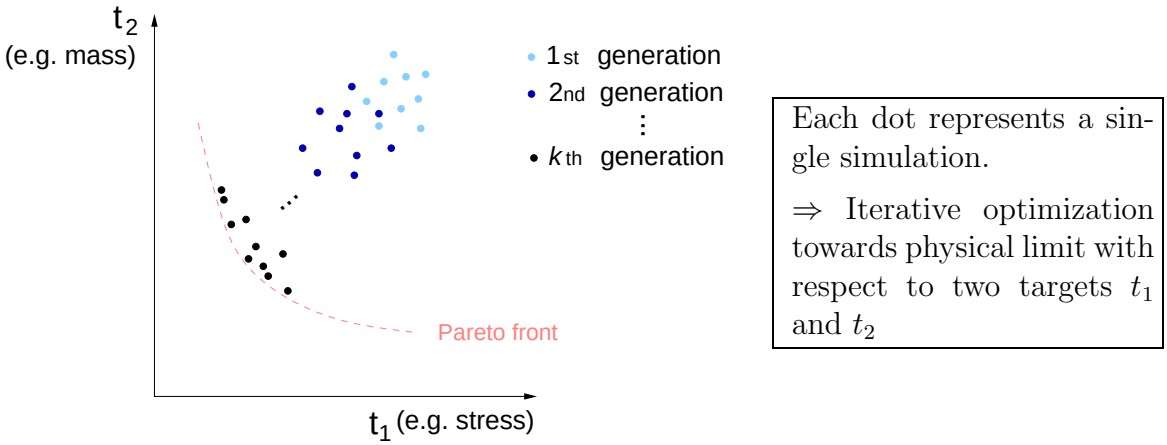


Figure 2.15: 2-D Scatter plot: Principle Idea of an Evolutionary Algorithm. Here: The population moves towards a physical limit, the Pareto front.

is seen in figure (2.16). According to table 2.1, we see that Run 1 requires an unnecessary amount of simulation runs due to slow movement through the design space. On the other hand, Run 2 very fast moves towards the design target but will not exhibit convergence behavior. This is due to the fact that deflections are too coarse to gather close enough around the design target. The proposed adaptive algorithm quickly moves towards the design target and then converges to the optimum parameter configuration. The chosen design in figure (2.16) for demonstrating the algorithm's convergence was the one with minimal Euclidean distance to the origin, giving a compromise design we assess as most useful.

	Standard deviation	Number of Shots N
Run 1	$\kappa\mu$	N
Run 2	$6\kappa\mu$	N
Adaptive Algorithm	$6\kappa\mu$, $3\kappa\mu$, $\kappa\mu$	N, 2N, 4N
μ :=mean value of distribution function;		$\kappa = 0.025$, N= 50

Table 2.1: Simulation parameters of three optimization runs

We recalculated the initial configurations $p^b_{1,1}$ and $p^c_{1,1}$ (cf. figures (2.17), (2.18)) with a small standard deviation. We chose 50 individuals in 60 generations. The initial configuration $p^b_{1,1}$ aborted the search process due to a hardly recognizable advancement in the search process. Here, the algorithm did not reach the Pareto front as individual $p^c_{1,1}$ finally does. Figure (2.19) shows the final designs from each of the three algorithms, giving clearly deviating configurations. We can here see the necessity to properly adjust ES parameters due to the problem at hand. More importantly, a criterion to decide whether

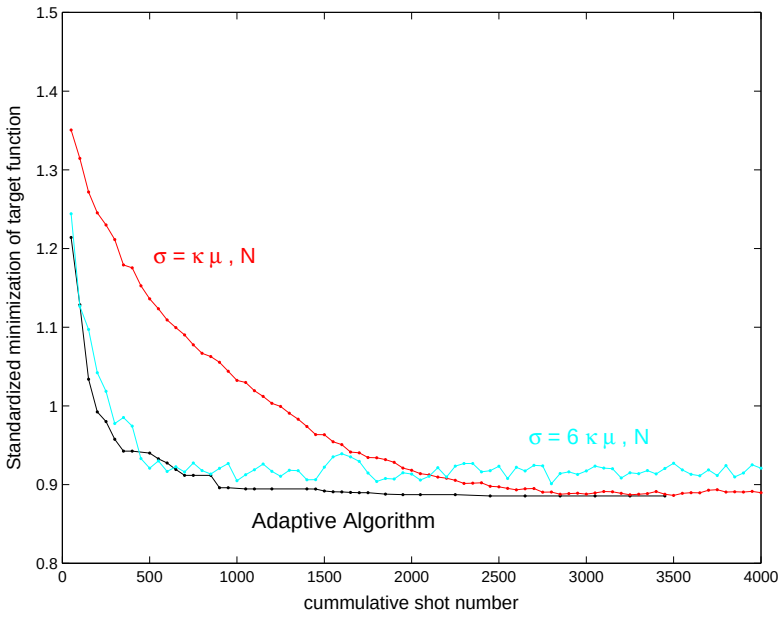


Figure 2.16: Comparison of 3 optimization runs with different parameter setup

an individual is approaching the Pareto front would be helpful. We will have solved the Pareto front approximation in the upcoming chapter.

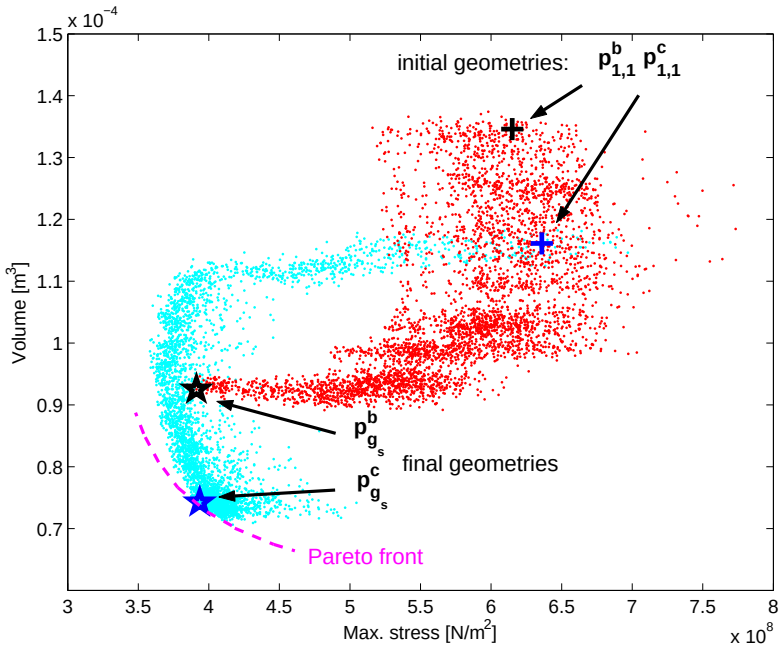


Figure 2.17: Scatter plot: Identical simulation parameters, varied initial geometries

Usually, a multi-objective optimization problem may be tackled by a weighted sum approach, such that weighting by constant factors $\mathbf{w} = (w_1, w_2)$ is introduced. In this way, it is for instance possible to find the best compromise of available weightings such that

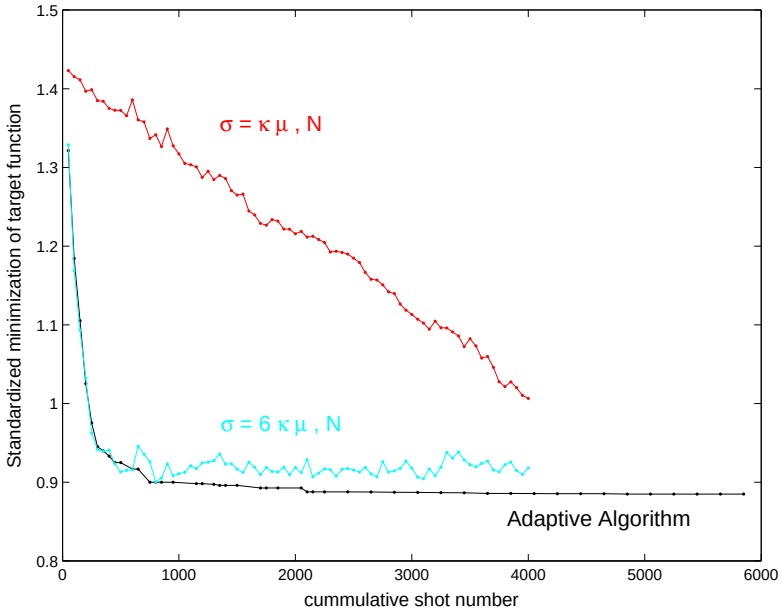
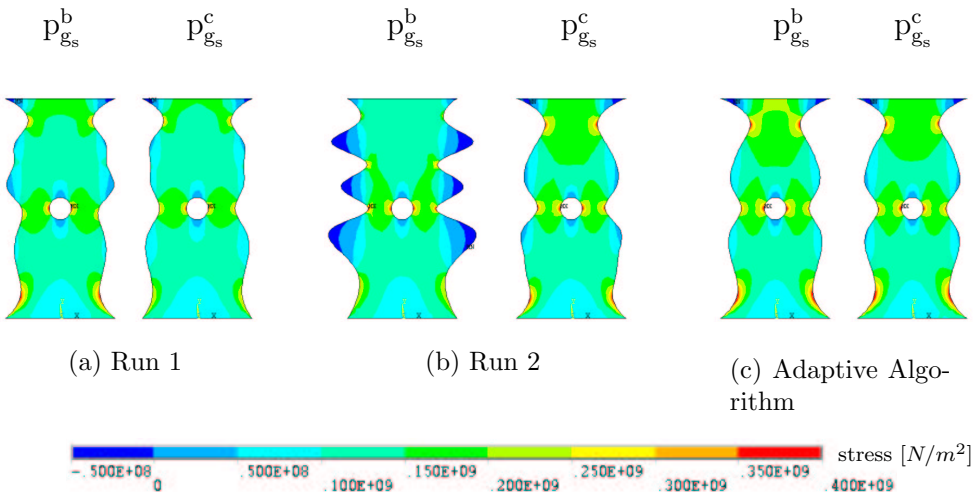


Figure 2.18: Optimization runs with different parameter setup (initial geometry $p^c_{1,1}$)



(a) Run 1 (b) Run 2 (c) Adaptive Algorithm

Figure 2.19: Derived geometries using different process configurations. We indicate the corresponding final designs.

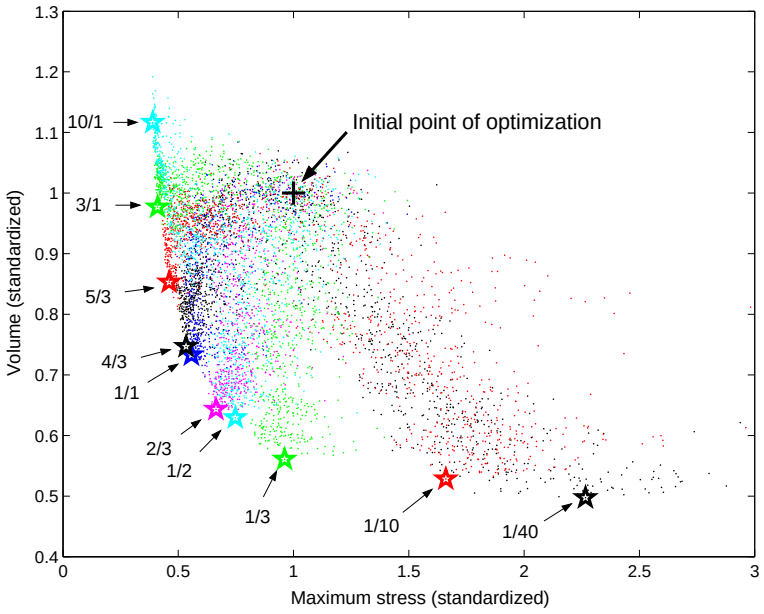


Figure 2.20: Weighted objectives to picture the Pareto front

the Euclidean distance of the objective is minimized. The fitness function then reads

$$f(\mathbf{w}, x_1, x_2) := \Big((w_1 t_1)^2 + (w_2 t_2)^2\Big)^{\frac{1}{2}}.$$

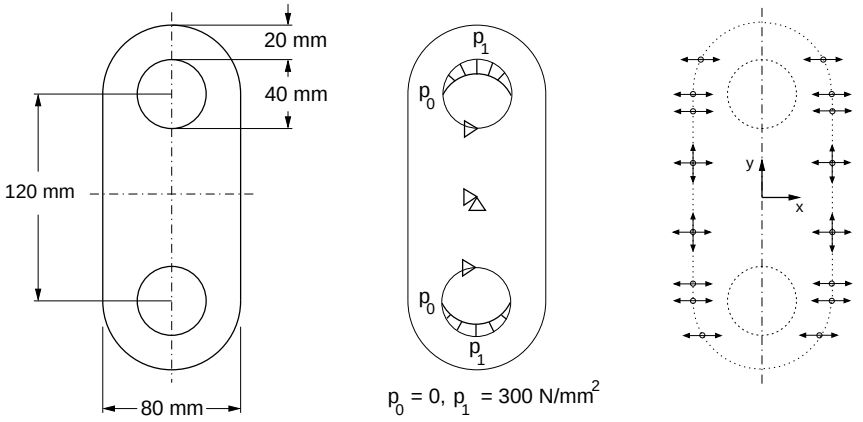
Here, we chose fixed weights $\mathbf{w}$ in the fitness function for each evolutionary strategy run involving 12 generations with 50 individuals. The process was then led to the corresponding region of the Pareto front exhibiting user preferences. Figure (2.20) gives the insight into design alternatives. The stars in figure (2.20) represent the target parameters in the final set of each run and the corresponding labels refer to the ratio w_1/w_2, clearly stating preferences for a trade-off between maximum stress and volume. We furthermore identify the physical limits for design configurations, the Pareto front. In this example the design engineer is able to pick a weighting corresponding to his subjective judgment.

Two more numerical examples will be shown to which the evolutionary algorithm is applied. The first example concerns a chain link which mass is to be reduced without stress increase. In the second example an analytically fully known case of a three-dimensional thin-walled tube under torsion is optimized towards minimum tip rotation for process validation.

2.7.2 Optimization of a chain link

Figure (2.21) indicates all necessary information by which the optimization task is set up. The material used is steel (Young's modulus E = $2.1 \cdot 10^{11}$ N/m^2, Poisson's ratio ν = 0.3) and a constant thickness (t = 5 mm) is assumed. Loads and boundary conditions

(BC) are applied according to figure (2.21 b). The definition of the design variables and hence the degrees of freedom (DOF) is shown in figure (2.21 c). Symmetry by the y-axis is enforced. The design target is to find a shape which has the minimum volume under the restriction that no increase in maximum stress occurs. Though this is a uni-modal optimization problem, we quickly and successfully find an optimal configuration. This optimization run involved 31 generations of 110 individuals.



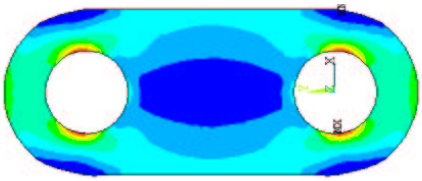
(a) Dimensions (b) BC (c) DOF

Figure 2.21: Chain link optimization setup

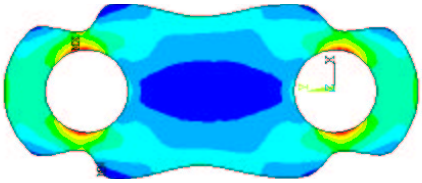
The results of the optimization and the final geometries are shown in figure (2.22). An improvement of the structure volume resp. mass of almost 8% was achieved with basically no increase in maximum occuring stress. Furthermore, the derived geometry is symmetric to both axes. This is confirmed by restarting the algorithm from several initial configurations all finishing with the same optimal shape design. Since the procedure was restarted from the boundaries of the design variable space, and also from a number of points in the inside of the decision space, we conclude that the functional coherence is rather smooth and uni-modal.

2.7.3 Optimization of a thin-walled tube

This example is based on a fully known mechanical background with given analytical solution. A thin-walled tube of which one end is clamped is loaded only by a torque M on its free end. The objective is to maximize the section modulus, which in turn means to reduce the deflection produced by the applied forces. Figure (2.23 a) shows the setup for this optimization task. The material chosen is aluminum (E=$0.706 \cdot 10^{11}$ N/m^2, $\nu = 0.345$) and the model consists of 11 relocatable vertices which define the bounding spline for each quarter of the tube cross section. The only degree of freedom (DOF) for these design variables is the corresponding radial direction. Symmetry to both cross sectional axes is enforced as depicted in figure (2.23 b). Additionally, at the given boundary conditions a restricted installation space (IS) is given, i.e. the evolving geometry is restricted to the



(a) Initial geometry



(b) Optimized shape

	stress $\sigma_{eqv,max}$ (10^8 N/m^2)	volume V (10^{-4} m^3)
(a)	0.621	0.604
(b)	0.627	0.556
Improvement		
	-0.1 %	7.9 %

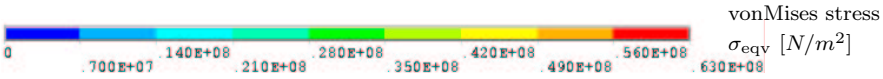
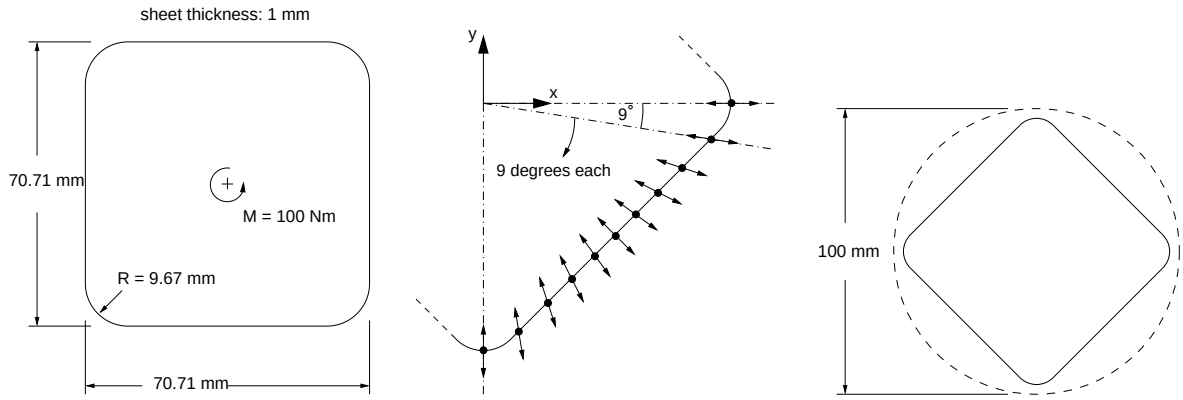


Figure 2.22: Results of the chain link optimization

inside of the dotted area as shown in figure (2.23 c). The total length is 500 mm. With



(a) Dimensions / load (b) DOF (c) IS

Figure 2.23: Example: Optimization of a thin-walled tube

respect to $t \ll r$, where t is the sheet thickness and r the tube's radius, a constant shear flow can be assumed in circumferential direction. The shear flow is defined to $n = \tau(s) \cdot t$ with $\tau(s)$ as shear stress depending on the boundary described (cf. [136,137]). According to Bredt's first formula the shear stress can be derived as:

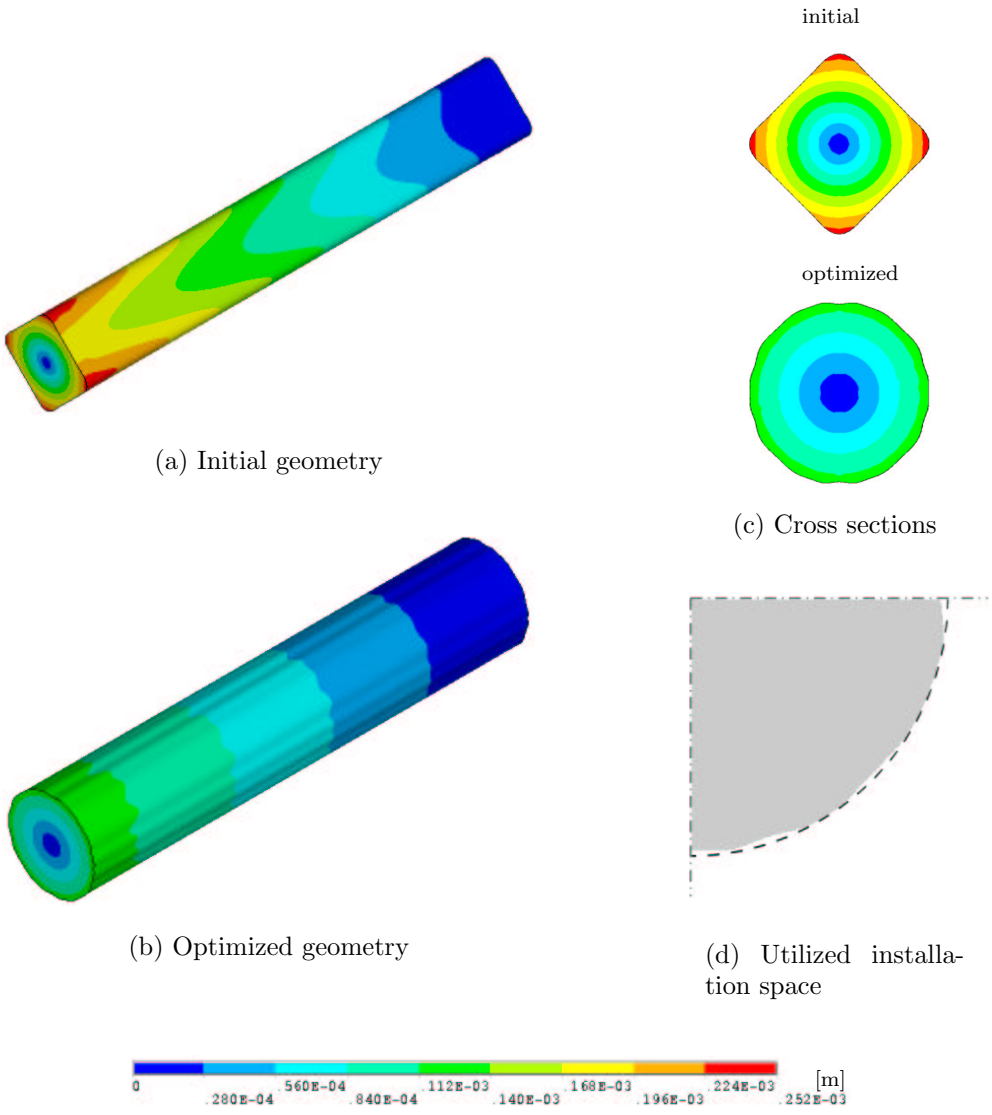
$$\tau(s) = \frac{n}{t(s)} = \frac{M}{2V} t(s).$$

Here, V denotes the design volume and M the outer torque. Hence the shear stress is maximal at the thinnest wall section. Since the sheet thickness is constant on the whole circumference in this example also the shear stress is constant. Thus, for the section modulus W we obtain

$$W = 2Vt.$$

Based on isotropic, linear elastic material laws for the displacement u follows

$$u \sim \frac{1}{W}. \tag{2.14}$$



(a) Initial geometry

(b) Optimized geometry

(c) Cross sections

(d) Utilized installation space

Figure 2.24: Optimization results: Thin-walled tube with torsion load

Equation (2.14) implies that with reducing maximum displacement, the section modulus (i.e. the cross-sectional area) has to be increased. Actually, this optimization task would

lead to an infinite enlargement of the cross section, i.e. the design variables would steadily move in the outward direction. By prescribing an installation space, this movement is restricted. The optimization aim is now to fully utilize the given space.

Figure (2.24) shows the initial configuration and the optimized design. The contours in figure (2.24 a-c) indicate the displacement which was efficiently reduced as expected. More interesting is the utilization of the given installation space. In figure (2.24 d) a quarter of the evolved geometry is displayed with the given space restriction described as a dashed line. One can see that the algorithm almost perfectly fills the installation space. The computations involved 31 generations with 80 individuals.

2.7.4 Conclusions

In this section, we demonstrated the performance of a heuristic optimization approach for structural mechanics shape design problems. We suggested an evolutionary strategy incorporating adaptive features. The well known drawback of these methods is a large number of function evaluations. The contemporary EA literature suggests to use several thousands of generations and individuals. On numerical examples we have seen that this drawback can be circumvented by carefully choosing ES parameters and the application of adaptive strategies. Moreover, evolutionary strategies offer to explore the Pareto front for multi-objective problems as we have shown. This gives the design engineer a number of alternative designs from which he may choose a proper one according to the desired operational set up.

The application of evolutionary strategies for engineering practice has shown to be powerful in accomplishing the optimization task. It is nevertheless necessary to apply methods suited for the particular optimization model. By considering that each function evaluation in linear elasticity only involves a few seconds, this cannot be said about the computationally more demanding flow evaluations. Since they take minutes up to hours for each function evaluation, one has to apply approximation models and special ES operators, respectively. The ongoing of this work will apply evolutionary strategy operators which specifically take these aspects and the efficiency of Pareto front approximation into account.

Chapter 3

Multi-Objective Optimization

"In practice, because of the changing nature of a problem, new constraints may make a previous optimum solution infeasible to implement. The knowledge of alternate optimum solutions allows an engineer to conveniently shift between one solution to another."

K. Deb [31]

In a multi-objective problem, we aim at determining an optimal vector $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_n(\mathbf{x}))$ which components are to be minimized according to certain criteria. We alter equation (2.1) to

$$\min_{\substack{x_l \leq x_i \leq x_u \\ i=1,\ldots,m}} \quad f_i(\mathbf{x}) : X \subset \mathbb{R}^m \mapsto Y \subset \mathbb{R}^n \quad i = 1, \ldots, n,\ n > 1,$$

where the inequality constraints are put into the decision space as to prevent the creation of invalid designs. This has practical reasons and will not mislead the following considerations.

As we argued in section 2.6, evolutionary algorithms are proposed to be suitable to render multi-objective problems. They work on a set of solutions and are hence applicable for this class of problems. These search strategies are subject to investigation for multi-objective optimization since more than a decade, a methodological and historical overview can be found in Deb et al. [31], Horn [70], and Zitzler et al. [169].

The dominance based Pareto front approximation proposed by Goldberg [53] has been taken up by numerous researchers, resulting in several Pareto based fitness assignment schemes and established the class of evolutionary multi-objective (EMO) algorithms. Nowadays, Pareto front approximation based on the dominance concept is seen as superior to sorting techniques. We pass on a literature presentation as the introduction of this work covered a contemporary overview.

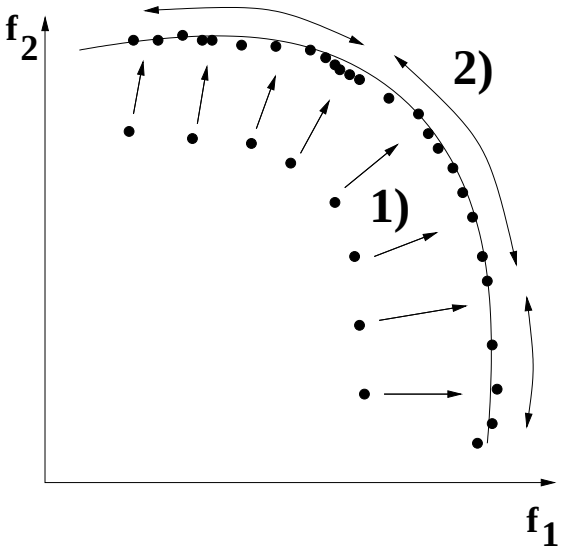


Figure 3.1: 1) Approximation to and 2) diversity on Pareto front

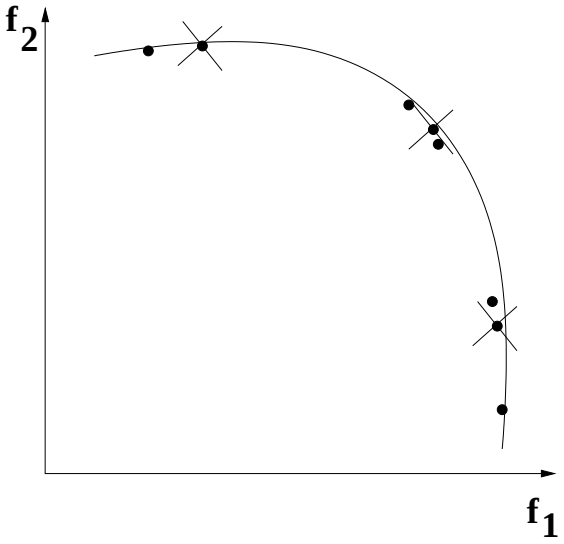


Figure 3.2: These three individuals are excluded from archive with size 5.

From the class of EMO methods, two algorithms are prevalent throughout the literature: The elitist non-dominated sorting genetic algorithm (NSGA-II) [27, 33] and the improved strength pareto evolutionary algorithm (SPEA2) [164, 165] both proved superior performance on a variety of test problems documented in, e.g., [26, 27, 29, 33, 162–165]. In these test problems, the objectives are analytical functions. Yet, these algorithms have not been used on engineering relevant applications which is also due to the fact that algorithms based on the Pareto dominance concept are anyway not well known in the engineering community. We also consider the simple evolutionary multi-objective optimizer (SEMO) and fair evolutionary multi-objective optimizer (FEMO) [91], less complex algorithms, yet still elitist strategies. As we focus on selection, the mutation and recombination operators are of secondary interest and will be explained at the end of this section.

Since there are no common metrices that allow a comparison between different algorithms for optimizing applications [168], we resort to judge algorithms by a skillful graphical illustration. The remarkable advantage from the application of the proposed methods is that all shape design alternatives become visible for the design engineer and a set of alternate solutions is gained in one optimization run, all in contrast to classical methods. The outcome of such a multi-objective optimization is a set of Pareto optimal solutions that visualize the trade-off between the competing objectives.

3.1 Evolutionary Multi-Objective Optimization

We have argued in the preceding sections that with multiple objectives to be optimized, we will hardly gain full information of the Pareto front with conventional single-objective algorithms. On the other hand, the variation, niching and selection operators in EA are

inherently designed to work on and manipulate solution sets. EA so has the potential to approximate the Pareto front in a complete simulation run. Pareto front approximation itself aims at two goals [31]: First, approximation to the Pareto front as close as possible and second, a good spread of solutions along the Pareto front (see figures (3.1), (3.2)). In both cases, operations on a set of solutions is demanded. Diversity along the Pareto front can be achieved by specific EA operators: The mutation and recombination operators ensure a sufficient diversity in the decision space while niching operators work in the objective space and are applied for simultaneously finding and describing several local extrema. These EA operators are slightly modified and are then used to lead the search towards exploiting the Pareto front: Mutation and recombination are responsible to drive the search towards the Pareto front, niching operators are used to spread along the Pareto front [68, 69]. These goals correspond to search and multi-criteria decision making as already mentioned in section 2.2.2.

The elitist feature guarantees that, due to the mutation from EA operators, the best solutions are not destroyed for diversity reasons. For a specific algorithm setup, convergence has been shown for elitist strategies. Unfortunately, they involve an arbitrarily number of generation cycles and are thus practically useless. In the following sections, we outline the optimization algorithms which found application in our numerical examples.

3.1.1 SEMO

Our first EMO algorithm is the conceptually simplest one, called simple evolutionary multi-objective optimizer (SEMO) [91,168]. SEMO consists of a population of variable size which stores all non-dominated solutions. A parent is drawn from the current population P according to a uniform probability distribution and subjected to a mutation (bit-flip) operator. Algorithm 1 details this process, where the symbol $>_P$ expresses the Pareto dominance relationship as defined in section 2.2.2. The generation loop aborts to a given number of generations.

3.1.2 FEMO

Next, the fair evolutionary multi-objective optimizer (FEMO) [91, 168] is introduced (see Algorithm 2). It uses an archive of variable size storing all non-dominated individuals. A counter $ct(i)$ is used storing the number of times an individual i has already been chosen for variation. The individual with the lowest count will be a parent individual in the next mating run. If there are several individuals with an equally low counter, one amongst them is randomly chosen. A new individual is added to the global population if it is not dominated by any other individual and if it is not equal in all objective values to any other individual in the archive. As in Semo, the algorithms iterates until a maximum number of generations.

Algorithm 1: SEMO main loop

1. a) Initialize algorithm, $P = \emptyset$

   b) Create an initial individual i (random starting point)

   $P = P \cup \{i\}$

   2. Generation loop

      Select one individual i out of P uniformly

      Create offspring i' by random bit mutation.

      $P = P \setminus \{k \in P \mid i' >_P k\}$

      if $\neg\exists k \in P$ such that $(k >_P i' \ \vee \ f_m(k) = f_m(i') \quad \forall m = 1, \ldots, M)$

      then

      $P = P \cup \{i'\}$

      end if

   end loop

3.1.3 NSGA-II

Deb et al. [27] proposed the non-dominated sorting genetic algorithm (NSGA-II) which demonstrated its superior performance on a number of test problems [33, 162, 163, 166]. Designed to be an elitist strategy, it preserves best solutions by the application of selection operators on the combined parent-offspring generation. The population size in NSGA-II is denoted by N, and each parent generation P_g has again to produce N offsprings (Q_g). The algorithm first combines the parent and the offspring population into one set $R_g = P_g \cup Q_g$ of size $2N$. After fitness assignment, this set is then ordered according to the Pareto dominance concept, i.e. first, an individual k is randomly chosen from the population R_g and inserted in an intermediate set. Then, another solution k' is drawn from R_g and compared to all individuals from the intermediate set. If k' dominates k, k' enters the intermediate population and k is deleted. The other way around, if k dominates k', then k' is deleted and k stays in the intermediate set. In this way, the intermediate population will consist of all non-dominated individuals of R_g, the Pareto front by definition. We will denote the intermediate set as the first Pareto front $\mathscr{P}_1$. $\mathscr{P}_1$ is then taken from the population and inserted into P_{g+1}. The remainder population is again subject to Pareto front identification, this time the Pareto front is denoted as $\mathscr{P}_2$, also inserted in P_{g+1}, and so on. During this process, the different Pareto fronts in the intermediate population R_g are sorted and denoted as $\{\mathscr{P}_i, i = 1, 2, 3 \ldots\}$. This procedure can be seen as a sorting based on the non-domination criterion of the population, hence the name non-dominated sorting genetical algorithm.

Algorithm 2: FEMO main loop

1. Initialize algorithm, $P = \emptyset$

   Create an initial individual i

   Set offspring counter to $ct(i) = 0$ and

   $P = P \cup \{i\}$

2. Generation loop

   Select one individual i out of $\{j \in P \mid ct(j) \leq ct(k)\ \forall k \in P\}$ uniformly

   Increment offspring counter $ct(i) = ct(i) + 1$

   Create offspring i' by random bit mutation.

   $P = P \setminus \{k \in P \mid i' >_P k\}$

   if $\neg\exists k \in P$ such that $(k >_P i' \ \vee \ f_m(k) = f_m(i') \quad \forall m = 1, \ldots, M)$

   then

   $P = P \cup \{i'\}$

   $ct(i) = 0$

   end if

   end loop

Therefrom, all Pareto fronts $\mathscr{P}_1, \ldots, \mathscr{P}_j$ are included in P_{g+1} as long as the whole front finds sufficient slots in generation $g + 1$. This process is continued until no more full Pareto front can be accommodated. Algorithm 3 details the Pareto front identification procedure. After that, the main algorithm activates the crowded sorting procedure based on a niching strategy. In the respective Pareto front, this routine calculates the distance of every individual to its neighbors, in each objective dimension, respectively (see figure (3.3)). The boundary solutions (solutions with smallest and largest function values) are assigned an infinite distance.

These cumulated distances are sorted in ascending order and thus give a hierarchy of crowding extent for each individual. A solution with a smaller value of this distance measure is more crowded by other solutions. Those individuals which are least crowded are first filled in the remaining slots of the new population. The following paragraph details the corresponding algorithm and definition. The main loop is described in algorithm 4.

Algorithm 3: Pareto front identification

1. $i = 1$
2. Apply non-dominated sorting to population and thus
   identify Pareto front $\mathscr{P}_i$ of population P_g
3. If $|P_{g+1}| + |\mathscr{P}_i| < N$ then
   $P_{g+1} = P_{g+1} \cup \mathscr{P}_i$
   else
   terminate and return i
   endif
4. $i = i + 1$
5. goto 2

Algorithm 4: NSGA-II main loop

1. g=0; Initialize parent population P_0 and offspring population Q_0 (Random Initialization)
2. Fitness Assignment of $P_g \cup Q_g$
3. Conduct Pareto front identification (Algorithm 3)
4. Conduct crowded selection operator (Definition 3.1.1) for Pareto front i
5. Include $(N - |P_{g+1}|)$ solutions into P_{g+1}
6. Termination: If stopping criterion ($g \geq G$) is satisfied then set $P_{final} = P_{g+1}$ and terminate.
7. Variation: Perform recombination and mutation operators to create new population Q_{g+1}.
8. $g = g + 1$
9. Goto Step 2

Crowded Tournament Selection Operator

Definition 3.1.1 below details the application of the crowded tournament selection operator. Before using this operator, we need to calculate the distances in the corresponding Pareto front. This done as in

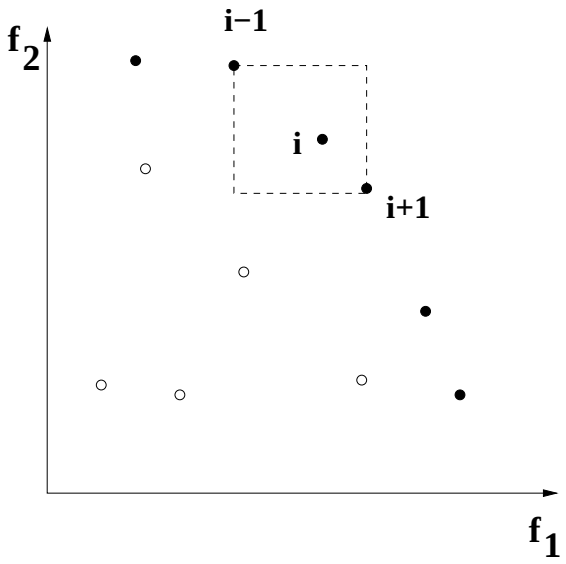


Figure 3.3: Distance assignment in NSGA-II

Algorithm 5: Distance Assignment

1. $l := |\mathscr{I}|$ denotes the size of the Pareto front $\mathscr{I}$. Set $d_i = 0$ for each $i \in 1, \ldots, l$.
2. Sort the set according to objectives f_m, $m = 1, \ldots, M$ via $I^m = sort(f_m, >)$.
3. For $m = 1, \ldots, M$

   set $d_{I_1^m} = d_{I_l^m} = \infty$, and

   for all $j = 2, \ldots, l-1$

   $d_{I_j^m} = d_{I_j^{m-1}} + \frac{f_m^{I_{j+1}^m} - f_m^{I_{j-1}^m}}{f_m^{max} - f_m^{min}}$

   end do

   end do

The index I_j^m denotes the solution of the j-th member on the m-th objective in the sorted list. Thus, for any objective, I_1^m and I_l^m denote the lowest and highest objective function values, respectively. The parameters f_m^{max} and f_m^{min} are set to be the maximum and minimum values of the m-th objective function.

Definition 3.1.1 *(The crowded tournament selection operator): An individual i wins a tournament against another individuals j if either one of the following conditions are true:*

1. *$r_i < r_j$, i.e. solution i has a better Pareto rank than solution j*
2. *$r_i = r_j$ and $d_i > d_j$, i.e. both solutions have the same rank but solution i has a better crowding distance than solution j.*

3.1.4 SPEA2

SPEA2 is an acronym for strength pareto evolutionary algorithm, proposed in Zitzler et al. [164,165] and was shown to perform well on several test problems. It is the so far most appropriate method for a variety of test problems [33,162,163,166]. In particular, SPEA2 is an improved version of the former SPEA [161].

The distinguishing feature of this algorithm is that it uses an external archive which stores a number of non-dominated individuals and is updated in each generation. SPEA2 is thus an elitist strategy preserving the best solutions over several generations. Futhermore, the algorithm utilizes an individual fitness assignment. Thus, solutions that belong to the first Pareto front $\mathscr{P}_1$ in the combined population are more appreciated in the selection procedure than any other solution. The fitness assignment is captured by using a raw fitness value that counts the number of individuals which are dominated. In addition, a density estimation is used to describe the individuals diversity performance. Adding density estimation and raw fitness together, we obtain the individual final fitness. Since the archive size is limited and fixed, it may happen that the size of non-dominated solutions exceeds the size of the archive. Then individuals are selected by a special clustering algorithm based on the k-th nearest neighbor method. This operator ensures that only the most valuable non-dominated individuals are preserved in the archive. The algorithm so steers the search process towards the most valuable regions in the search space.

We denote the population size by N, the archive size by N_{arc}, the external archive by P^*, and G is the number of generations. The algorithm output is the nondominated set P_{final}. We first describe the main algorithm loop in algorithm 6, the proceeding will then detail the corresponding operators.

3.1.4.1 Fitness Evaluation

We identify each point in the objective space $\mathbf{x} \in \mathbb{R}^n$ as an individual i. The population is so represented in the objective space where the selection operators apply. First, each

Algorithm 6: SPEA2 main loop

1. Initialization: $g = 0$, initialize population P_0 (Random Initialization) and set external archive $P_0^* = \emptyset$.
2. Fitness Assignment: Determine fitness for all individuals $P_g \cup P_g^*$ (see section 3.1.4.1)
3. Archive Update (detailed in section 3.1.4.2):

   Copy all nondominated individuals in $P_g \cup P_g^*$ to P_{g+1}^*.

   If $|P_{g+1}^*| > N_{arc}$ *then* apply truncation operator.

   If $|P_{g+1}^*| < N_{arc}$ *then* fill P_{g+1}^* with dominated individuals. aber falsch! (6.10.04)
4. Termination: If stopping criterion ($g \geq G$) is satisfied then set $P_{final} = P_{g+1}^*$ and terminate.
5. Selection: Perform tournament selection with replacement on P_{g+1}^* to fill the mating pool.
6. Variation: Perform recombination and mutation operators to the mating pool and set P_{g+1} to the resulting population.
7. $g = g + 1$
8. Goto Step 2

individual $i \in P_g \cup P_g^*$ in the population and the archive is assigned a strength value

$$S(i) = |\{j \mid j \in P_g \cup P_g^* \ \wedge \ i >_P j\}|$$

used in the raw fitness calculation

$$R(i) := \sum_{j \in P_g \cup P_g^*,\ j >_P i} S(j).$$

$R(i) = 0$ corresponds to a non-dominated individual whereas a high value of $R(i)$ means that i is dominated by many individuals which again dominate many other individuals.

The density estimation technique in SPEA2 is an adaption of the k-th nearest neighbor method, where the density at any point is a decreasing function of the distance to the k-th nearest data point. For each individual i we proceed as in algorithm 7.

3.1.4.2 Archive Update

The external archive is updated as follows. The first step is to copy all nondominated individuals into the archive of the next generation:

$$P_{g+1}^* = \{i \mid i \in P_g \cup P_g^* \ \wedge \ F(i) < 1\}.$$

Algorithm 7: SPEA2 density assignment

1. Distances for each i in the objective space to all individuals j in archive and population are calculated and stored in a list.
2. The list is sorted in increasing order.
3. The k-th element gives the sought distance, denoted as σ_i^k ($k = \sqrt{N + N_{arc}}$).
4. the density $D(i)$ is then defined by

$$D(i) := \frac{1}{\sigma_i^k + 2},$$

   where the 2 in the denominator ensures that $0 < D(i) < 1$.

5. The final fitness $F(i)$ of an individual i is calculated as

$$F(i) = R(i) + D(i).$$

Three cases may appear, depending on the archive size: If the archive is exactly as large as the first Pareto front (i.e. the number of non-dominated individuals), the whole Pareto front is copied into the archive. Otherwise,

Algorithm 8: SPEA2 archive update

If $|P_{g+1}^*| < N_{arc}$ *then*

1. Sort $P_g \cup P_g^*$
2. Copy $N_{arc} - |P_{g+1}^*|$ (dominated) individuals from this list to P_{g+1}.

If $|P_{g+1}^*| > N_{arc}$ *then* employ archive truncation:

Remove individual $i \in P_{g+1}^*$ for which $i \leq_d j$ for all $j \in P_{g+1}^*$.

This means that the solution which has the minimum distance to all other solutions is chosen. If there are several solutions with minimum distance, second smallest distances are considered:

$$\begin{aligned} i \leq_d j \quad :\Longleftrightarrow \quad & \forall\, 0 < k < |P_{g+1}^*| :\ \sigma_i^k = \sigma_j^k \ \text{ or} \\ & \exists\, 0 < k < |P_{g+1}^*| :\ \left[\left(\forall 0 < l < k :\ \sigma_i^l = \sigma_j^k\right) \ \text{ and } \sigma_i^k < \sigma_j^k\right], \end{aligned}$$

where σ_i^k denotes the distance of i to its k-th nearest neighbor.

Recombination and Mutation

One-bit mutation is the most commonly used mutation operator. In this case, one entry in the chromosomal string is switched according to a certain probability. Independent-bit mutation randomly flips every bit in the chromosomal string.

In recombination, a crossing point is chosen according to a certain probability distribution here, two binary strings swap their substrings. This is known as one-point crossover. Uniform crossover determines for each bit position separately whether the value is copied from parent one or parent two.

3.2 Numerical Example I

As the first test problem we consider the multi-objective optimization of a heat exchanger configuration with respect to pressure drop, maximum temperature, and fluid area. A heat exchanger is a device in which energy is transferred from one fluid to another across a solid surface. Some examples are intercoolers, preheaters, boilers and condensers in power plants. After describing the problem and the corresponding flow model we investigate the performance of the proposed method and the influence of its characteristic parameters.

3.2.1 Problem Description

The considered heat exchanger configuration consists of 8 hot pipes at temperature $T_\mathrm{h} = 400°\mathrm{C}$ passed by a fluid within a rectangular box with inlet and outlet channels. The configuration is illustrated in figure (3.4) together with the numerical grid. The ambient temperature is $T_\mathrm{u} = 290°\mathrm{C}$ which is prescribed at the inlet and the outer walls. The flow is defined to be water and the fluid properties are chosen such that the Reynolds number based on the inlet velocity and channel height is $Re = 40$. In this, the dynamic viscosity is $\mu = 1 \cdot 10^{-3}$ Pas, the density $\rho = 1000$ kg/m^3, thermal capacity $c_p = 4000$ J/kgK and the coefficient of thermal conductivity is $\lambda = 0.6$ W/mK, giving the Prandtl number $Pr = 6.70$.

The objectives to be optimized are the pressure drop between the inlet and the outlet channel, the maximum temperature across the outlet, and the total area occupied by the fluid. The eight design variables are the diameters of the pipes. There is a strong interaction between the objectives, such that the problem is representative for a conflicting multi-objective optimization. In particular, the selected problem appears to be well suited to investigate issues like:

- How do, according to different optimal solutions, design alternatives actually look?
- How can one validate or falsify claims about coherence of objectives?
- How can one choose alternatives or compromises between objectives?

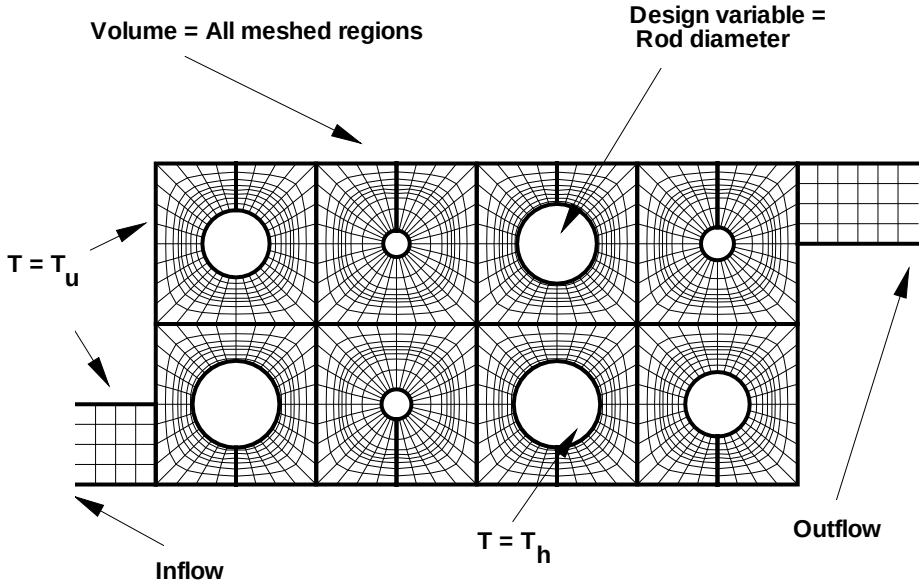


Figure 3.4: Cutout of Flow Geometry

- What are the trade-offs between the objectives, i.e. how much do we have to sacrifice of one objective to gain improvement according to another objective?

We will address these questions on the basis of the results obtained.

3.2.2 Discretization

The spatial discretization employs 35 328 control volumes on the finest grid and 5 grid levels are used in the multi-grid method (in figure (3.4) the third grid is shown). The geometry representation is parametrized with respect to the pipe diameters and the mesh is adapted automatically in a corresponding way. The pipe diameters are restricted to upper and lower limits such that there can not occur invalid shapes in the decision variable space.

3.2.3 Numerical Results

The main optimization computing environment is provided by the PISA library [168]. For this example, we applied the strength Pareto evolutionary algorithm. SPEA2 is applied and implemented according to its description in the cited literature [164, 165] and has been modified to suit our purposes. In contrast to the algorithm outline in the preceding section, the k-th nearest neighbor density estimator the parameter $k = 1$ is chosen here for efficiency reasons.

We divide the numerical experiments in two scenarios, each of which entails several simulation runs. In the first scenario, we chose a high probability for individuals to undergo

mutation and recombination. This means that the population rapidly changes its chromosomal attributes such that a high diversity bias is achieved. The second scenario slows down this effect until population diversity is only driven by recombination. The settings for the two scenarios are summarized in table 3.1.

Scenario	Mutation type	Mutation/Bit-turn probabilities	Recombination type	Recombination probability
1	independent-bit	0.5/0.3	Uniform crossover	0.9
2	no mutation	0/0	one-point crossover	0.7

Table 3.1: Scenario Simulation Settings

In each scenario, we vary the population and offspring size as well as the number of generations. This is because each fitness evaluation represents a computational demanding numerical flow solver run. So we are interested to seek a visible Pareto front with as few flow solver runs as possible. The simulation configurations are summarized in table 3.2, where also the correspondence with the scatter plots (figures (3.5) to (3.48)) is indicated.

No.	Number of generations	Initial population	Number of offsprings	Function evaluations	Tournament size	Corresp. figures
1-1	20	180	38	1194	6	3.5-3.8, 3.9-3.12
1-2	40	20	20	1113	6	3.13-3.16, 3.17-3.20
1-3	20	40	20	539	3	3.21-3.24, 3.25-3.28
2-1	18	150	36	590	6	3.29-3.32, 3.33-3.36
2-2	40	20	20	554	6	3.37-3.40, 3.45-3.46
2-3	19	40	20	290	6	3.41-3.44 , 3.47-3.48

Table 3.2: Parameters for simulation runs for scenarios 1 and 2

In each scatter plot, we indicate vectors a_1, a_2, a_3 and b_1, b_2, b_3, which correspond to the maximum a_i and minimum b_i of each objective $i = 1, 2, 3$, respectively. This gives a good impression of the design alternatives. The corresponding temperature distributions are given in the figures indicated in table 3.2. For sake of clarity, the graphical illustrations are summarized on the consecutive pages following these explanations.

The conflict between objective functions pressure drop and temperature increase can clearly be seen in different designs. The Pareto fronts between fluid area and pressure drop as well as between pressure drop and temperature are clearly visible. In the following, we describe these effects on each of the corresponding simulation runs. Conclusion are aggregated at the end of this chapter.

Figure (3.5) is the three-dimensional plot between pressure drop, volume and temperature. Figures (3.6)-(3.8) are the respective two-dimensional views on the plot (3.5). In order to clarify the view on our results, we keep this split-up of two and three dimensional views throughout this chapter. The mentioned plots show a three dimensional Pareto front arising between all objective functions. Figure (3.6) does actually not show a Pareto front, we will see in later examples, that this is due to the utilized optimization scenario.

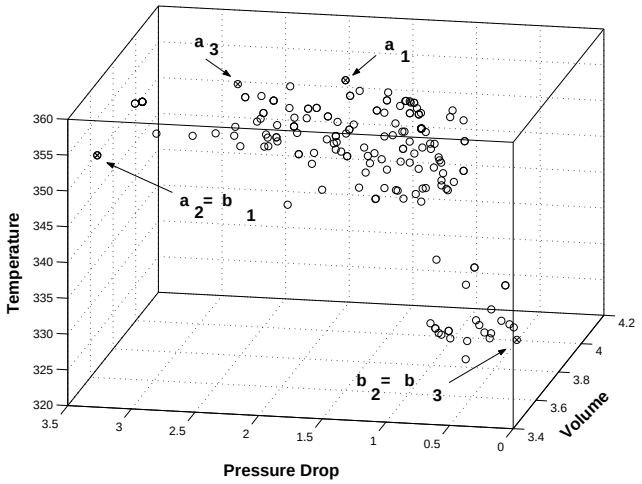


Figure 3.5: 3-D scatter plot, scenario 1/1

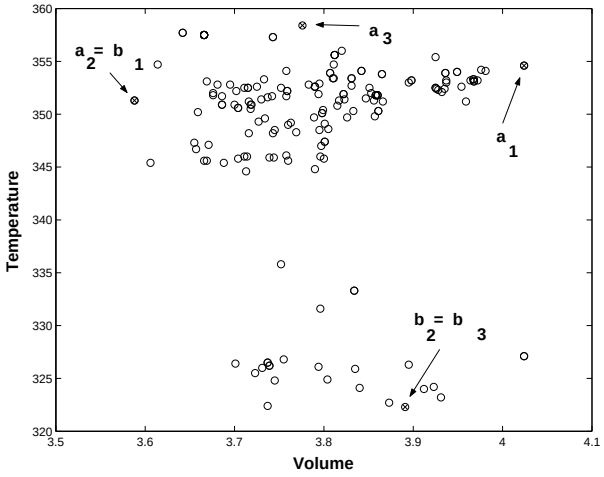


Figure 3.6: Scatter plot surface area-temperature, scenario 1/1

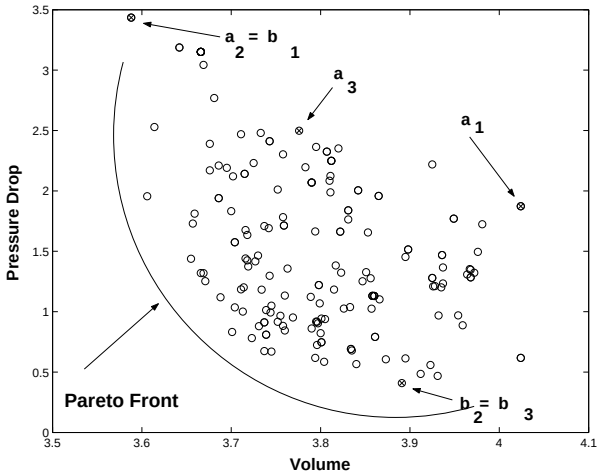


Figure 3.7: Scatter plot surface area-pressure drop, scenario 1/1

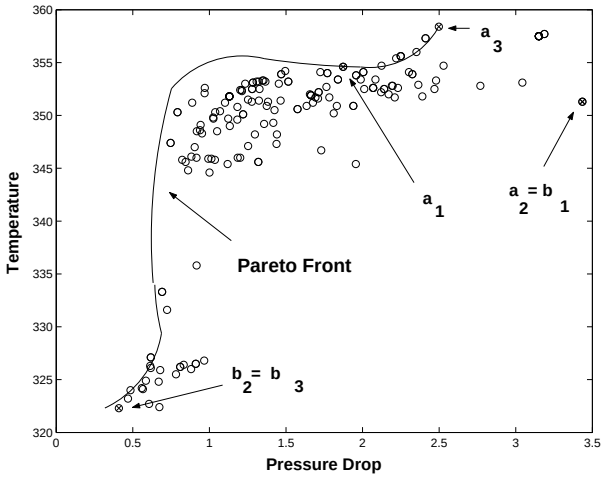


Figure 3.8: Scatter plot pressure drop-temperature, scenario 1/1