DISS. ETH NO. 17017

Representation Concepts in Evolutionary Algorithm-Based Structural Optimization

A dissertation submitted to the
Swiss Federal Institute of Technology
Zurich

for the degree of
Dr. sc. ETH Zurich

presented by
Mathias Giger
Dipl. Masch. Ing. ETH
born on August 31, 1977
citizen of Niedergösgen (SO)

accepted on recommendation of
Prof. Dr. P. Ermanni, examiner
Prof. Dr. E. Zitzler, co-examiner

2007

Abstract

This thesis investigates several aspects of numerical optimization in the world of structural engineering. In particular the representation of structural parts as a prerequisite for any optimization process is of great importance. Typically, the representation intensively interacts with the applied optimization algorithm and decisively contributes to the generation of superior design solutions. Moreover, the qualities of the optimization algorithm also heavily influence the convergence behavior of the optimization process.

The development of CFRP racing motorcycle rims is presented, whereas EAs and a constant-length vector genotype are applied during the development process. The application of optimization methods provides optimum stacking sequences which lead to decisively lighter rear and front rims compared to state-of-the-art magnesium alloy rims. The identified needs for further research in terms of the performance of the optimization algorithm and the representation concept are addressed in this thesis.

AORCEA - an Adaptive Operator-Rate Controlled Evolutionary Algorithm - is developed which changes the operator rates of the variation operators based on the search state of the optimization algorithm. A success and a relative diversity measure are applied which are used as an indicator for the adjustment of the variation operator rates. AORCEA is tested on several well-known numerical benchmark functions as well as on the optimization of a steel trellis motorbike frame. The adaptive strategies always perform superior or at least equal to a reference EA with constant variation operator rates.

In particular in the field of composite laminate optimization the representation of the stacking sequence is a critical issue. A patch concept treating single laminate layers as design entity is the basis for a variable-length representation concept. The optimization of a simple eigenfrequency problem indicates that this variable-length representation concept may lead to superior solutions with less evaluations. Furthermore, the optimization of a hockey stick is presented as an example for an engineering application.

Besides the traditional vector representation also mathematical graph-based representation concepts are investigated within this thesis. The optimization of truss structures is investigated due to the obvious similarity between trusses and mathematical graphs. The nodes and members are encoded as vertices and edges of a graph, respectively, and a major advantage is the independency from ground structures which are the basis of many of the popular truss optimization methods. Moreover, an important advantage of graph-based representations is the possibility of concurrent optimization of topology, shape, and sizing what today is most often strictly separated.

The graph-based approach leads to novel solutions which hardly could be found with traditional methods.

A graph-based representation concept is also developed for global laminate optimization. Such laminate structures are most often evaluated by FE-simulations. The underlying FE-mesh which typically consists of layered shell elements is ideally suited for a representation concept. The finite elements are represented in a mathematical graph which further groups them to zones. The stacking sequences of all zones are optimized and the zone layout allows for local reinforcements and the incorporation of domain knowledge.

Modern CAD systems are essential for product development processes because they offer a parametric-associative representation of mechanical structures. A generic EC framework is established which provides all required functionalities to optimize topology and shape of such mechanical structures. The CAD package CATIA V5 and its C++ interface CAA V5 are ideally suited for the implementation of an optimization framework because the mechanical structures are represented by specification trees. This specification tree is interpreted as mathematical graph which allows for traditional shape but also topology optimization, i.e., the generation of structures with a variable number of design entities is rendered possible.

For all these graph-based representation concepts appropriate variation operators are developed and the concepts are validated with illustrative applications which lead to convincing results.

Zusammenfassung

Diese Dissertation befasst sich mit verschiedenen Aspekten der numerischen Optimierung im Bereich der Strukturoptimierung. Eine besondere Bedeutung kommt dabei der Repräsentation der Strukturen zu, die eine wichtige Voraussetzung für den Optimierungsprozess darstellt. Sie interagiert mit dem Optimierungsalgorithmus und trägt entscheidend zu der Entwicklung von bedeutend besseren Optimierungslösungen bei. Zusätzlich hat auch die Qualität der Optimierungsalgorithmen einen bedeutenden Einfluss auf das Konvergenzverhalten der Optimierung.

Der Entwicklungsprozess von faserverstärkten Karbonfelgen, die mit Evolutionären Algorithmen und Vektorgenotypen konstanter Länge optimiert worden sind, wird in dieser Arbeit vorgestellt. Die Anwendung der Optimierungsmethode liefert einen optimierten Lagenaufbau, der zu Hinter- und Vorderfelgen führt, welche bedeutend leichter sind als die Standardfelgen aus Magnesiumlegierungen. Die erkannten weiteren Forschungbedürfnisse in Bezug auf die Leistung des Optimierungsprozesses und der Repräsentationskonzepte werden in dieser Dissertation thematisiert.

AORCEA - der *Adaptive Operator-Rate Controlled Evolutionary Algorithm* - wird entwickelt, der die Anwendungswahrscheinlichkeiten der Variationsoperatoren basierend auf dem momentanen Suchverlauf steuert. Ein Erfolgs- und ein relatives Diversivitätsmesskriterium werden als Indikator für die Anpassung der Anwendungswahrscheinlichkeiten der Variationsoperatoren verwendet. AORCEA wird anhand von verschiedenen numerischen Testfunktionen sowie mit der Optimierung eines Stahlrohrmotorradrahmens überprüft. Die adaptiven Strategien erbringen jeweils eine bessere oder zumindest gleich gute Optimierungsleistung wie der Referenzalgorithmus mit unveränderlichen Anwendungswahrscheinlichkeiten der Variationsoperatoren.

Im Gebiet der Laminatoptimierung stellt die Repräsentation des Lagenaufbaus eine besondere Herausforderung dar. Ein *Patchkonzept*, welches eine einzelne Laminatlage als kleinste Einheit betrachtet, bildet die Basis für ein Repräsentationskonzept mit variabler Länge. Die Optimierung eines einfachen Eigenfrequenzproblems indiziert, dass dieses variabel lange Repräsentationskonzept zu überlegenenen Optimierungslösungen führen kann bei deutlich geringerem Evaluationsaufwand. Die Optimierung eines Eishockeystockes dient schliesslich als Beispiel um die Anwendbarkeit dieses Repräsentationskonzeptes für Ingenieurprobleme zu demonstrieren.

Neben der traditionellen Vektorrepräsentation werden auch auf mathematischen Graphen basierende Repräsentationskonzepte entwickelt. Die Optimierung von Fachwerken wird untersucht, da eine offenstichtliche

Ähnlichkeit zwischen mathematischen Graphen und solchen Fachwerken besteht. Die Gelenke und Stäbe werden als Knoten beziehungsweise Kanten eines Graphen repräsentiert. Ein grosser Vorteil dieser Methode ist die Unabhängigkeit von einer Grundstruktur, die die Grundlage vieler bekannter Fachwerkoptimierungsmethoden darstellt. Zudem stellt die gleichzeitige Optimierung von Topology, Form und Grösse (Querschnitte der Stäbe) einen weiteren wichtigen Vorteil der Methode dar, da diese Aspekte bei anderen Methoden häufig getrennt betrachtet werden müssen. Das graphbasierte Repräsentationskonzept führt zu neuartigen Lösungen, die mit herkömmlichen Methoden kaum hätten gefunden werden können.

Ein graphbasiertes Repräsentationskonzept wird auch für die Optimierung von Laminatstrukturen entwickelt. Laminatstrukturen werden häufig mit FE-Simulationen berechnet und das zugrundeliegende FE-Netz, welches normalerweise aus Schalenelementen mit Lagenaufbau besteht, ist bestens als Basis für ein Repräsentationskonzept geeignet. Die finiten Elemente werden in einem Graph abgebildet der diese wiederum zu Zonen gruppiert. Der Lagenaufbau all dieser Zonen wird optimiert wobei die Anordnung der Zonen lokale Verstärkungen und das Einbringen von Ingenieurwissen ermöglicht.

Moderne CAD Systeme sind aus der modernen Produktentwicklung nicht mehr wegzudenken da sie eine parametrisch-assoziative Darstellung der mechanischen Strukturen ermöglichen. Eine generische Optimierungsumgebung, die alle Funktionalitäten für Topologie- und Formoptimierung bietet, wird entwickelt. Die CAD Software CATIA V5 und das integrierte C++ Interface CAA V5 ist sehr gut für die Implementation einer solchen Optimierungsumgebung geeignet, weil die mechanischen Strukturen in einem Spezifikationsbaum repräsentiert sind. Dieser Spezifikationsbaum wird als mathematischer Graph interpretiert, der sowohl normale Form- als auch Topologieoptimierung zulässt, d.h. die Generierung von Strukturen mit einer unterschiedlichen Anzahl Bausteinen wird ermöglicht.

Für alle diese graphbasierten Repräsentationskonzepte werden passende Variationsoperatoren entwickelt. Die Tauglichkeit dieser Konzepte wird jeweils mit anschaulichen Beispielen, die zu überzeugenden Resultaten führen, unter Beweis gestellt.

Acknowledgements

This thesis was accomplished at the Centre of Structure Technologies of the Swiss Federal Institute of Technology in Zurich. The presented work was supported by *Kommission für Technologie und Innovation* (KTI Nr. 6469.1) and the Swiss National Foundation (SNF-project 200020-103861/1).
I would like to thank:

- Professor Paolo Ermanni and Dr. Gerald Kress for giving me the opportunity of working in this interesting field of research and their guidance during this time.
- Professor Eckart Zitzler for being co-examiner of my thesis.
- Oliver König, Nino Zehnder, and Marc Wintermantel for introducing me into the fascinating world of structural optimization and supporting me the last few years.
- David Keller for his contributions to this work and the excellent proofreading.
- all the people at the Centre of Structure Technologies, particularly René Roos and Florian Hürlimann for being humorous office mates.
- Matthias Kaufmann, and Cornel Boesch for their diploma and semester theses which contributed to this work.
- my parents and family for their lifelong support and encouragement.
- Maja for everything.

List of Symbols

A	Adjacency matrix, 191
a	Cross-sectional area, 108
$\mathbf{a}$	Vector of cross-sectional areas, 108
D	Rating for objective and constraints, 30
d	Euclidean distance, 63
ϵ	Uniform mutation range, 180
F	Fitness function value, 30
$\mathbb{G}$	Genotype space, 24
γ	Random proportional factor, 29
$\mathbf{i}^*$	Current best individual, 35
$\mathbf{i}$	Individual (decision vector), 30
O	Design objective value, 30
$\mathbf{o}$	Offspring solution, 29
P	Population size, 6
$\mathbb{P}$	Phenotype space, 24
$\mathbb{P}_{feas}$	Feasible solutions in the phenotype space, 25
$\mathbb{P}_{infeas}$	Space of infeasible solutions, 25
$\mathbf{P}$	Population of individuals, 35
$\mathbf{p}$	Parent solution, 29
σ	Standard deviation, 180
v	Current gene value, 180
w	Weighting factor, 30
x	Solution in the genotype space, 24
x'	Solution in the phenotype space, 24
y	Solution in the genotype space, 24
y'	Solution in the phenotype space, 24

List of Acronyms

ABG	Adaptive Biological Growth, 97
AORCEA	Adaptive Operator Rate Controlled Evolutionary Algorithm, 27
bff	Best fitness frequency, 60
BGL	Boost Graph Library, 192
C++	Object-oriented programming language, 12
CAA V5	Component Application Architecture Version 5, 12
CAD	Computer Aided Design, 1
CAO	Computer Aided Optimization, 3
CEA	Coevolutionary Algorithms, 7
CFD	Computational Fluid Dynamics, 1
CFRP	Carbon Fiber Reinforced Plastics, 11
DOF	Degree of freedom, 186
EA	Evolutionary Algorithms, 5
EC	Evolutionary Computation, 2
EOlib	Evolving Objects library, 36
EP	Evolutionary Programming, 6
ES	Evolution Strategies, 6
ESM	Element Stiffness Matrix, 184
ESO	Evolutionary Structural Optimization, 97
FEM	Finite Element Method, 1
FSD	Fully Stressed Design, 97
GA	Genetic Algorithms, 6
GP	Genetic Programming, 6
GPS	Generative Part Structural Analysis workbench, 158
GSM	Global Stiffness Matrix, 184
GUI	Graphical User Interface, 173
IFS	Iterated Function Systems, Fractal Theory, 9

LISP	Computer programming language, 7
MOEA	Multi-objective evolutionary algorithm, 30
MP	Mathematical Programming, 2
OCM	Optimality Criteria Method, 2
PARDISO	Parallel Sparse Direct Linear Solver, 187
RNG	Random Number Generator, 103
SAGA	Species Adaptation Genetic Algorithm, 79
SKO	Soft Kill Option, 3
SO	Shape Optimization, 9
STL	Standard Template Library, 193
TOD	Topological Optimum Design, 9
UDF	User-defined feature, 150
UED	Uniform Energy Distribution, 97
VLG	Variable-length genotype, 12

Contents

Chapter 1

Introduction

This dissertation investigates several aspects of numerical optimization in the world of structural engineering. In particular the representation, also called parameterization or encoding, of structural parts as a prerequisite for any optimization process is of great importance. Typically, the representation intensively interacts with the applied optimization algorithm and decisively contributes to the generation of superior design solutions. Moreover, the qualities of the optimization algorithm also heavily influence the convergence behavior of the optimization process. Only an adequate combination of representation scheme and optimization algorithm guarantees for sophisticated design solutions at lower development costs.

This introductory chapter presents the state-of-the-art of structural optimization which meanwhile has become an enormous field of research. First, in Section 1.1 different approaches to structural optimization are discussed and a thorough literature review is given. The subsequent Section 1.2 presents the goals of this thesis emerging from the identified needs for further research. Finally, a detailed overview of the thesis is given in Section 1.3.

1.1 State-of-the-art of structural optimization

Nowadays, product development processes are inseparably linked to modern software technologies in order to virtually design, analyze, and manage products from the first conceptual ideas to the manufactured end products. During the entire design process computers come into operation by application of a variety of design and simulation software. In the beginning basic design concepts, also called topologies or layouts, are developed. Subsequently, preliminary designs specifying different shapes and, finally, detailed models with exact geometrical specifications are created. The Computer Aided Design (CAD) technology is often employed to generate digital and fully-parametric design models allowing for the comparison of different design variations. Furthermore, these models are used as a basis for numerical

simulations, for instance by the Finite Element Method (FEM) predicting the structural behavior or by Computational Fluid Dynamics (CFD) simulations quantifying aerodynamic properties. It is self-evident that a lot of numerical optimization strategies are directly based on such software tools.

In the field of structural optimization a variety of different automated optimization methods have been investigated and applied during the last decades of research. A popular way to categorize these methods is to use their optimization algorithms as distinguishing feature. Three main categories can be identified:

- *Mathematical Programming* (MP)
- *Optimality Criteria Methods* (OCM)
- *Evolutionary Computation* (EC)

The following subsections shortly explain the characteristics of MP and OCM – they are sometimes called *formal methods* – and mention some interesting applications. The subsection concerning the EC methods is more detailed since this thesis is exclusively concerned with methods from this category.

1.1.1 Mathematical Programming

The methods of MP generally require continuous objective and constraining functions. Further requirements with respect to differentiability need to be fulfilled depending on the MP method at hand, *Direct* (zeroth-order) methods only require objective function values, whereas *gradient-based* (first-order) methods also need the first derivative. *Second-order* methods are additionally based on the Hesse matrix containing the second derivatives. Apart from the method order, search methods are also characterized by the model order which is the order of the Taylor series expansion that approximates the objective function. Some examples of established algorithms can be given. A direct method with model order zero is the *Simplex Search Method* [1], and the *Powell's Conjugate Direction Method* [2] and the *Response Surface Method* [3] are examples for direct methods of model order two not requiring any derivatives of the objective function. Examples for first-order methods are the *Cauchy Method* [4] of model order one and the *Fletcher-Reeves Conjugated Gradient Method* [5] of model order two. Finally, a second-order approach of model order two is the *Newton Method* [6].

Typically, a formal model of the optimization problem needs to be developed that resembles the original problem close enough. This approach quite often requires simplifications, whereas oversimplifications may lead to solutions which do not solve the original problem. MP is well suited for convex objective functions since most of the algorithms are able to nearly

or exactly locate the global optimum. In case of non-convex objective functions only local optima can be obtained and multi-start techniques need to be employed in order to increase the probability of discovering the global optimum. Unfortunately, the MP methods with high model order become inefficient with increasing number of design variables. As an example, for the Newton Method the number of function evaluations to calculate the Hesse matrix scales quadratically with the number of design variables so that for a large number of design variables the numerical effort becomes too high.

Probably the most popular development based on MP methods in the field of structural optimization is the *homogenization method* initiated by Bendsøe and Kikuchi [7, 8, 9]. Their method is suited for topology optimization (cf. Section 2.1) and is directly based on a Finite Element mesh of a fixed design domain. Each Finite Element of the design domain has to be either void or full of material resulting in a discrete optimization problem. The originally discrete topology optimization problem is rendered continuous by introducing a continuous density function for each element, hence, the optimal material distribution can be determined by using gradient information. The density of each element can vary between zero and the density of the massive material. These intermediate density values are then mapped to the element material properties so that the optimization process converges to solutions having a discrete 0-1 material distribution. The homogenization method was originally developed to solve the convex and differentiable minimum compliance problem subject to a mass constraint. However, the method has been extended to allow for anisotropic material models [7, 10], frequency optimization [11], and combined shape optimization strategies [12]. Several commercial software packages provide topology optimization implementations based on the homogenization method, e.g., *TOSCA.topology* [1], *ANSYS* [2], and *Optistruct* [3].

Although the MP methods can be efficient for continuous and convex problems, they cannot cope with typical structural optimization problems which are most often of discrete and highly non-convex nature. Moreover, a common disadvantage of all FE-based structural optimization methods is that their results directly depend on the granularity of the FE-discretization. Therefore, a compromise has to be found between a coarse discretization to keep the number of design variables moderate and a fine mesh to achieve sufficient simulation accuracy.

1.1.2 Optimality Criteria Methods

A different approach to structural optimization are the so-called Optimality Criteria Methods (OCM) [13]. They are based on a local optimality cri-

[1] http://www.fe-design.de
[2] http://www.ansys.com
[3] http://www.altair.com

terion, e.g. stress level, which has to be fulfilled in any region of a given structural part. These Finite-Element based methods work locally by modifying appropriate nodes or elements of the FE-mesh and are therefore mainly suited to minimize local stress concentrations. In general, these approaches are often computationally efficient because no global objective has to be evaluated and the convergence behavior is independent from the number of design variables. The most popular implementations, i.e. the Soft Kill Option (SKO) and the Computer Aided Optimization (CAO), were introduced by Mattheck [14]. Both methods use an analogy to biological growth strategies of trees or bones. The SKO is based on the assumption that the stiffness of the design will globally increase if the Young's modulus is increased in highly stressed and reduced in less stressed regions. If the stresses fall below a given threshold value, Young's modulus is reduced to zero in order to completely remove the material. The CAO method is directly inspired by the natural growth of trees, where material is accumulated at heavily loaded locations in order to reach a homogenization of the failure probability over the entire structural part. Several commercial software distributors have implemented these methods for shape and sizing optimization (cf. Section 2.1), e.g., *TOSCA.shape* and *MSC.Nastran Optimization* [4].

A major drawback of OCM lies in the nature of its local approach. In general, only local modifications of structural parts can be found and the incorporation of additional constraints, e.g. maximum allowable stresses or compliance measures, is difficult.

1.1.3 Evolutionary Computation

The EC methods are stochastic (zeroth-order) search algorithms modeling the natural phenomenon of evolution, i.e., a combination of the Darwinian concept of *"survival of the fittest"* and the inheritance of genetic material within a certain species. They are heuristic in nature and generally cannot guarantee to find a global optimum solution. However, they are applicable to even highly complex models for which standard optimization methods, e.g. gradient-based algorithms, are not applicable and in practice they often yield promising results. The terminology of EC is inspired by the strong resemblance to biological processes, i.e., lots of the terms are borrowed from genetics and cellular biology. A candidate design solution is called an *individual* and a set of such solutions is called *population*. Furthermore, the representation (parameterization, encoding) of an individual is called its *genome* or *genotype* consisting of a sequence of *genes*. For some algorithms the search space (*genotype space*) is explicitly separated from the solution space (*phenotype space*) and an additional *mapping* procedure is required to *code* (phenotype to genotype space) or *decode* (genotype to phenotype

[4] http://www.mscsoftware.com

space) the candidate solutions. The process modifying individual solutions to produce new solutions is known as *breeding* or *mating* and the newly created individuals are called *offspring*. A *fitness* value is assigned to each individual indicating its quality in the context of the given problem in order to compare different candidate solutions. During the optimization process environmental selection is imposed in order to determine which of the parents and the offspring survive and are transfered to the new population which is typically called a new *generation*.

EC is an enormous field of research concerned with the application of Evolutionary Algorithms (EA) to complex real-world optimization problems. Probably the most recent literature review is presented by Kicinger [15], but also some older surveys [16, 17] provide an insight into the world of EC. First, a basic introduction to EAs is given, followed by an overview of applications in the field of structural optimization.

1.1.3.1 Evolutionary Algorithms

The canonical Evolutionary Algorithm Although many different implementations of EAs exist, the basic concept of the so-called canonical EA forms the basis of all of them. The canonical EA consists of the steps presented in Algorithm 1 and can be understood as a search process in which a population experiences gradual changes.

Algorithm 1 Canonical EA

```
1: t = 0
2: Initialize the population
3: Evaluate the complete population
4: while (not termination condition) do
5:    t = t + 1
6:    Mating selection
7:    Apply variation operators
8:    Evaluate offspring
9:    Select individuals for survival
10: end while
```

Before the evolutionary process starts, an initial population of a given number of individuals needs to be created at generation t = 0. Typically, this initialization process is of completely random nature, but also other initialization techniques incorporating knowledge of the optimization problem, e.g. the insertion of already known solutions into the starting population, are frequently applied. Next, each individual needs to be evaluated with respect to the predefined quality measures and a typically scalar fitness value is assigned.

After these preparatory steps the time counter is incremented and the

actual evolutionary process begins by selecting a subset of the current population as parents for the generation of new individuals. The mating selection process favors individuals having higher fitness values over candidate solutions with below-average fitness values. This holds for maximization problems, whereas for minimization problems a lower fitness value indicates better qualities of the candidate solution. Afterwards, new individuals are created by copying them and applying genetic variation operators. The most popular types of genetic variation operators are *mutation* and *recombination*. Usually, mutation operates on a single individual by changing at least one gene value of its genotype, whereas recombination acts on several (typically two) individuals by recombining their genotypes. The newly generated individuals obviously need to be evaluated and a fitness value is assigned. Subsequently, either all individuals or only a subpopulation of the current population are selected for survival in the selection step. In case all individuals are replaced the algorithm is called *generational* and the replacement of subpopulations occurs in *steady-state* algorithms. The steps 5 to 9 of Algorithm 1 are performed until a given termination condition, e.g. a maximum number of generations or evaluations or a target fitness value, is met.

Evolutionary-Algorithm paradigms Today, four main EA paradigms are accepted in the research community, namely, Genetic Algorithms (GA), Evolutionary Programming (EP), Evolution Strategies (ES), and Genetic Programming (GP). The different branches have rather historical roots, but more and more the differences between these paradigms blear and a lot of popular EC applications cannot be clearly associated to one of these paradigms. The four branches are presented next in their pure forms as described in the first publications.

Genetic Algorithms were originally developed by Holland [18, 19] and subsequently studied by Goldberg [20], De Jong [21], Davis [22], and Grefenstette [23] to name only a few. The genotype space is binary ($\{0,1\}^n$) and the phenotype space can be any space, as long as it can be coded into bitstring genotypes. The mating selection scheme is proportional, i.e., each individual has a probability proportional to its fitness to be selected for reproduction. The only crossover operator is the so-called *one-point crossover* which replaces a segment of bits of the first parent with the corresponding segment of the second parent. Furthermore, the uniform mutation operator is applied which randomly flips bits of the parent with an user-defined probability and in the generational selection stage all offspring replace all parent solutions.

Evolutionary Programming, introduced by Fogel [24, 25] and extended by his son David Fogel [26] and others [27, 28], was originally devel-

oped as an attempt to create artificial intelligence. This paradigm clearly emphasizes the phenotype space and simply relies on mutation operators without applying classical mating selection. Every individual in the current population of size P generates exactly one offspring, whereas the best P individuals among parents and offspring become individuals of the next generation.

Evolution Strategies were initially designed with the goal of solving difficult discrete and continuous parameter optimization problems by Rechenberg [29] and Schwefel [30] and were then further investigated by many others, e.g. Bäck [31]. The original (1+1)-ES algorithm consists of a population holding a single individual defined as a real-valued vector. The only variation operator is *Gaussian mutation* where a zero-mean Gaussian variable with standard deviation σ is added. Then, the parent and the offspring individual are compared and the better candidate solution becomes the parent of the next generation. The choice of the parameter σ is obviously critical. Thus, the so-called 1/5 thumb rule is applied to adjust the parameter σ: If more than $\frac{1}{5}$ of the mutations yield to improved solutions σ is increased, otherwise σ is decreased.

Current state-of-the-art ES approaches [32, 33] are population based, i.e., the basic concept is extended to so-called $(\mu + \lambda)$-ES and (μ, λ)-ES algorithms, where μ parents generate λ offspring individuals and numerous further mutation operators exist.

Genetic Programming is the most recently developed paradigm invented by Koza [34]. It is a method for evolving computer programs [35] which can be represented as tree-like structures, e.g. LISP (list processing) programs. The optimization algorithm itself is similar to the classical GA with respect to initialization and selection. The main difference lies in the tree-like representation and the customized variation operators. Crossovers typically exchange subtrees to generate new offspring individuals and mutation operators replace arbitrarily chosen subtrees with randomly initialized tree structures. The fitness evaluation of newly created individuals is typically performed by running the program which is represented by the tree-like genotype, whereas the fitness depends on how efficient the program solves a given problem.

Advanced EA methods The four above-mentioned paradigms are presented in their original specification, but all of them have been further developed with respect to each step presented in Algorithm 1. Novel approaches for initialization, mating selection, variation operators, and selection strategies enhance the capabilities of these different search algorithms. Moreover, several hybrid algorithms which cannot be classified by the presented paradigms have been developed. They combine several

different features from at least two of the paradigms or they even combine EC methods with MP approaches [36]. Also the attention paid to *multi-objective* EAs remarkably increased during the last two decades. Instead of searching a single optimum solution, they seek for a set of alternative trade-offs also known as *Pareto-optimal* solutions [37, 38, 39, 40, 41, 42]. A further variation of the basic EA concepts are *parallel* algorithms evolving optimum solutions in multiple parallel subpopulations, whereas some individuals occasionally migrate to other subpopulations [43]. The so-called *coevoultionary algorithms* (CEAs) also rely on multiple subpopulations but additionally modify another fundamental assumption, namely that individuals are no more evaluated independently of one another. Two popular models are the *cooperative* CEA [44], where the fitness of a candidate solution is evaluated through 'cooperation' with individuals from other subpopulations, and the *competitive* CEA [45], where the fitness assignment is based on a competition of individuals of different subpopulations.

Field of application Due to their stochastic nature EC methods are excellently suited to tackle highly non-convex optimization problems where gradient-based methods normally fail to produce significantly improved solutions. Nevertheless, they are typically more expensive in terms of function evaluations compared to MP or OCM strategies, i.e., whenever possible some formal methods should be used. On the other hand, the evolutionary paradigm is a 'weak' method making only a few assumptions about the problem domain, hence, it can be applied to almost any optimization problem.

In general, EC can be implemented with different levels of knowledge inclusion. The optimization problem can be transformed into a suitable form to apply a standard algorithm, or a customized *evolutionary system* is developed which fits as close as possible to the characteristics of the phenotype space. In other words, it has to be decided how much domain knowledge is incorporated into the evolutionary system in terms of representation, variation operators, and the corresponding objective function. Although there is no general answer to the question of how much domain knowledge should be included, many practical applications and experimental evidence indicate that as much domain knowledge inclusion as possible leads to more successful evolutionary optimizations. Thus, an evolutionary system typically consists of a basic EA which is adjusted to the needs of the data structure to be optimized [46], i.e., the representation of the individuals as well as the appropriate variation operators are problem-dependent.

1.1.3.2 Applications in the field of structural optimization

The mentioned applications are grouped into the three major problem categories existing in the field of structural optimization:

- *Topological optimum design* (TOD), is also simply called topology optimization searching for an optimal material distribution in a given design domain,
- *Shape optimization* (SO) seeks for the optimal shape or contour of a structural system with given topology, and
- *Sizing optimization* looks for optimal dimensions, e.g. cross-sectional areas, of a structural system with fixed topology and shape.

Topological optimum design Typically, TOD problems are divided into *continuum* and *discrete* TOD groups. Continuum TOD is based on a discretized design domain consisting of usually rectangular (FE-)elements, where each element contains material or is void. This problem formulation is known from the homogenization method (cf. Section 1.1.1) and an EC approach is presented by Sandgren et al. [47] and Jensen [48]. In their formulation the mass of the structure is minimized by means of a GA and displacement as well as stress constraints are incorporated. Further research on this subject is presented by Chapman and Jakiela [49, 50] and Kane et al. [51, 52] particularly addressing the problem of unconnected sub-domains of the resulting mechanical structures and disruptive crossover operators leading to offspring which is not similar to neither of its parents. More advanced concepts overcome the known drawbacks of FE-based parameterization, e.g., mesh-dependent results or checkerboard patterns, by proposing the use of Voronoi diagrams, hole representations, and IFS representations based on fractal theory [53]. In [54] the Voronoi-based parameterization is even applied on a multiobjective topology optimization problem.

Discrete TOD addresses the determination of the optimal element connectivity from a finite number of possible connections and is typically applied to truss structures. The initial problem formulation for linear programming from Dorn et al. [55] established the so-called *ground structure approach* providing a basic set of potential connections. It is suited for small problems, but as soon as the problem size and the number of design variables increase and due to the discontinuous nature of the problem formal methods may become inefficient. Thus, the application of GAs to this problem class is a logical step and in particular Hajela [56] presents sophisticated solutions to several problems. Rajan [57] concurrently optimizes topology, shape, and member sizing of truss structures based on the ground structure approach and Rajeev and Krishnamoorthy [58] uses a variable-length string representation. A real-valued representation has been recently presented by Azid et

al. [59] and a demonstrative example for an optimization of steel structures can be found in Erbatur [60]

The emergence of novel materials and process technologies in the field of structural design has also great influence on the development of structural optimization methods. The outstanding mechanical properties and the functional features offered by composites, which are manufactured from fiber-reinforced plies, demand for optimization methods able to exploit the full potential of these materials. The optimal placement of laminate plies (material) can also be considered as a kind of topology optimization. Hansel et al. [61] describe a heuristic optimization which is based on local criteria in order to find weight-minimal laminate structures. Besides some gradient-based optimization methods of laminated structures, e.g. Zowe et al. [62], which hardly can cope with the discrete nature of such optimization tasks, also EC methods are applied. The use of a GA for the minimum thickness design of composite laminated plates is explored by Le Riche et al. [63]. Soremekun [64] adapts the representation of a laminated panel structure to concurrently optimize the number of layers as well as their orientation and also Nagendra et al. [65] and Kogiso et al. [66] present a stacking sequence optimization. All these approaches have in common that they optimize the stacking sequence of a panel or a plate, but they offer no possibility to change the shape of certain laminate plies to produce local reinforcements of the structure. For a review of further applications of composite optimizations see also Venkataraman [67].

Shape optimization Shape optimization generally maintains a fixed topology but changes the shape of a structural part. Analogous to the TOD case, shape optimization can be divided into groups of continuum and discrete SO. Continuum SO deals with 2D and 3D structures which have oriented boundary curves or surfaces, respectively. Formal methods are well established for solving continuum SO problems [68, 69], but also a series of EC methods have been developed. In [70] a concept for GA-based shape optimization is presented which moves the boundary nodes of a FE-model to develop the optimal shape for the given problem. Jenkins [71] performs research on the shape optimization of structural members and an optimization of a heterogeneous flywheel is presented by Huang et al. [72].

The discrete SO methods are concerned with variations of the geometry of truss and frame structures by changing the locations of the connecting nodes [73]. An application based on EC methods has been conducted by Grierson and Pak [74] optimizing the shape of frame structures.

Sizing optimization Sizing optimization is generally easier compared to TOD and SO because it only aims at finding optimal moments of inertia or dimensions of structural elements. This kind of optimization problem

is relatively well understood and the application of formal methods [75] is favorable as long as the objective function is convex. The cross-sections of members of a plane 10-bar truss are optimized by Goldberg and Samtani [76] using GAs. Hajela [77, 78] thoroughly investigates cross-section optimization of discrete member trusses and Deb [79] applies GAs to optimize designs of welded beams.

1.2 Goals of the thesis

In the beginning of this thesis the development of CFRP (carbon fiber reinforced plastics) racing motorcycle rims has been carried out (c.f. Chapter 3). During this work some major drawbacks of the traditional representation of composite laminate structures have been found as well as insufficient optimization performance due to inefficient application of variation operators has been detected.

In particular for structural optimization problems the evaluation of a single individual can be very expensive, i.e., an evaluation can take minutes or even hours. It is therefore of high importance to apply most efficient optimization algorithms providing improved solutions with few evaluations. The development of adaptive optimization algorithms is therefore a promising approach to increased the overall optimization performance.

Furthermore, the rim optimization was run with constant-length genotypes only circumstantially allowing for a changing number of laminate plies what is an absolute prerequisite for mass-saving optimizations. The introduction of variable-length representation concepts is a further development of the parameterization of laminated structures which also contributes to the optimization performance since non-coding regions are omitted in the genotype. During the work on variable-length representations the idea for a mathematical graph-based representation concept emerged which overcomes some drawbacks of the variable-length vector genotype representation, e.g., the suboptimal mapping procedure from single laminate plies to the underlying FE-model. In the course of this development the graph-based representation which implicitly provides variable-length representation properties turned out to be suitable for a wide field of topology optimization problems, e.g., optimization of truss structures or CAD-based topology optimization. A major part of this thesis is therefore dedicated to the development of graph-based representation concepts for structural optimization problems.

The main goal of this thesis is to develop an EC-framework which is suited for structural optimization, whereas a strong emphasis is laid on the representation of topology and laminate optimization problems. Three major subgoals can be identified:

- The first goal of this thesis directly addresses the identified drawbacks of the CFRP motorcycle rim optimization. A variable-length representation giving the opportunity of easily use a changing number of design entities is introduced and an adaptive EA is developed which is able to change the operator rates according to the search state in order to increase the optimization performance.

- The second goal is to develop novel representation concepts based on mathematical graph theory which are suited for truss as well as for laminate optimization. Mathematical graphs implicitly allow a variable size of the representation but they also require the development of suitable variation operators as they are known from GP applications. An important advantage of such representations is the possibility of concurrent optimization of topology, shape, and sizing what today is most often strictly separated. However, only a concurrent optimization of all of these aspects allows for the exploitation of the full potential of design improvement.

- Finally, the third goal aims at adapting the mathematical graph-based representation approach to make it usable for direct CAD optimization. Today, only simple shape (parameter) optimization is state-of-the-art in the established CAD software packages what can be overcome by using a graph representation. The C++ interface CAA V5 of the CAD system CATIA V5[1] is employed to directly apply the variation operators on the tree-like specification of an arbitrary mechanical part. Thus, concurrent topology, shape, and sizing optimization of fully-parameterized CAD models is rendered possible.

1.3 Thesis outline

The thesis is organized as follows: Chapter 2 presents an EA which is adapted to the special requirements of structural optimization. In particular the representation of individuals, the design of variation operators, and the definition of fitness functions are discussed. The development of CFRP racing motorcycle rims is presented in Chapter 3 as 'starting point' of this thesis highlighting again the needs for further research. The following Chapter 4 addresses the investigation of an adaptive EA whose operator rates are controlled based on an analysis of the convergence behavior. All further chapters are then concerned with the representation issue in the field of structural optimization. Chapter 5 pursues a straight-forward approach to

[1] http://www.ibm.com/catia

introduce more flexible representations by allowing for variable-length genotypes (VLG). Then, Chapter 6 demonstrates a mathematical graph based representation of truss structures which is probably the most simple application in structural optimization due to the obvious similarity between trusses and mathematical graphs. Besides the optimization of truss structures, the mathematical graph representation is also excellently suited for the representation of laminated composite structures. The necessary adaptations and the appropriate variation operators are presented in Chapter 7 which is concluded with the optimization of a sailplane wing structure. Finally, Chapter 8 deals with the development of CAD-integrated topology, shape, and sizing optimization which is also based on a mathematical graph representation. The thesis is concluded with Chapter 9 summarizing the findings and giving an outlook for further research in the enormous field of structural optimization.

Chapter 2

Evolutionary Algorithms in structural optimization

There are many different implementations of EAs which are tailored to the respective optimization problem. While some researchers still rely on the traditional GA approach with bit string representation, others trust in real-valued or even heterogeneous representations [80] in order to adapt the EA to the optimization problem. Apparently, the evolutionary process works for all of these representation types. Generally, the criteria which must be met for evolution to occur are: *reproduction*, *inheritance*, *variation*, and *selection*. As stated in Bentley [81]:

> "..., as long as some individuals generate copies of themselves which inherit their parents' characteristics with some small variation, and as long as some form of selection preferentially chooses some of the individuals to live and reproduce, evolution will occur."

Some attempts were conducted to theoretically investigate the evolutionary mechanism reliably producing improved solutions. On the one hand, Markov chain analyses are available for the standard bit string GA [82]. On the other hand, the so-called *Schema Theorem* [19] and the *Building Blocks Hypothesis* [20] are often cited to explain the convergence of, at least, canonical GAs. A *schema* is a template identifying a subset of bit strings with similarities at certain string positions and the schema theorem states that short, low-order, schema with above-average fitness increase exponentially in successive generations. In addition the building-block hypothesis states that the short, low order, and highly fit schema are sampled, recombined, and resampled to form strings of potentially higher fitness. These particular schema are named building blocks, whereas the hypothesis suggests that GAs are able to evolve improved solutions by combining these above-average fit schema to form superior bit strings.

With these principles in mind, the development of a customized EA matching to the requirements of structural optimization, particularly in terms of representation and variation operator design, can be approached.

First, the different problem categories of structural optimization are presented which heavily influence the design of EC methods. Then a basic EA scheme is outlined which is used within this thesis and the purpose of each step is briefly explained. Afterwards, important issues concerning the representation, the variation operator design, and the fitness function definition are discussed in detail. Finally, some remarks on the implementation and a survey of the representation and adaptation concepts developed within this thesis conclude the chapter.

2.1 Problem categories in structural optimization

Particular requirements to EAs in terms of structural optimization can be identified by considering the different types of optimization of a mechanical structure. In general, three different problem categories, namely topology (TOD), shape (SO), and sizing optimization as already defined in Section 1.1.3.2 and illustrated in Figure 2.1 for continuum and in Figure 2.2 for discrete structures, are distinguished. In most of the published optimization problems one category is addressed isolated from the others, i.e., either the topology, the shape, or the sizing is optimized. Further applications follow a sequential approach, where in different design stages of the development process a tailored optimization strategy is employed. The concurrent optimization of all three aspects of a mechanical structure is rare because some optimization methods are only suited for a single problem category, e.g. the homogenization method, or it is a too complex task. However, one might argue that only the concurrent optimization can lead to a superior solution since, e.g., a sequential optimization of topology and shape only can find the optimal shape for the previously developed topology, whereas a concurrent optimization would provide the best combination of topology and shape.

Rozvany [83] defines *layout* optimization as the simultaneous selection of the optimal topology, geometry, and cross-sectional dimensions in the context of truss optimization. This concept can be generalized for structural optimization of arbitrary mechanical parts where geometry is replaced by shape and sizing stands for the cross-sectional dimensions.

Evolutionary Computation is perfectly suited to perform layout optimization which can be considered as the most general approach to structural optimization. In order to pursue the layout optimization approach, the representation and the variation operators need to be designed in a way that layout optimization is possible.

After the presentation of a basic EA scheme, the aspects of representation and variation operator design are discussed in detail in Sections 2.3

and 2.4.

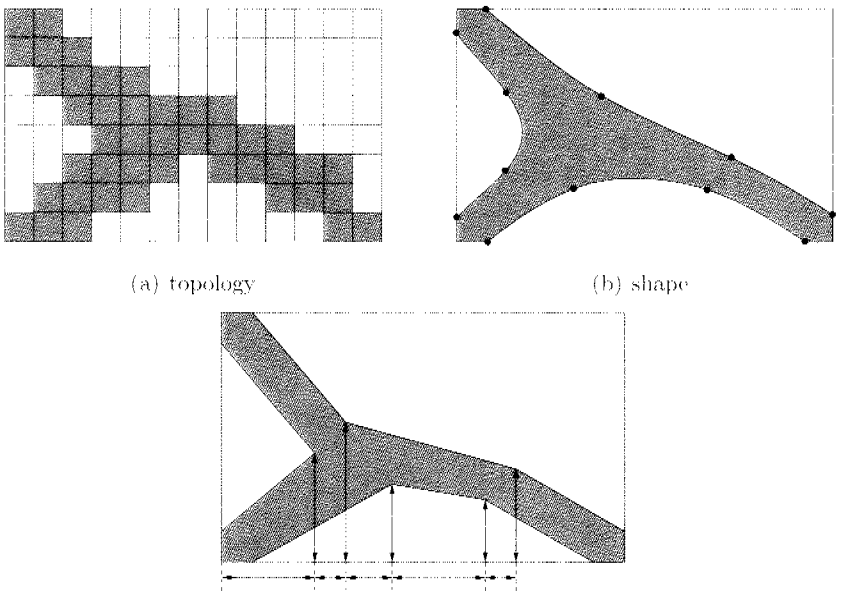


Figure 2.1: Problem categories for continuum structures.

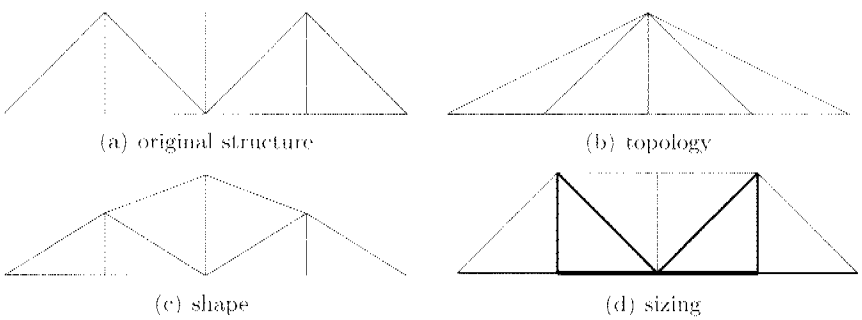


Figure 2.2: Problem categories for discrete structures.

2.2 A basic EA scheme for structural optimization

Figure 2.3 shows the basic EA scheme used within this thesis. The EA scheme is similar to the basic Algorithm 1 presented in Section 1.1.3.1 except for the mapping and the adaptation stage. All the shaded stages are typically problem-dependent, i.e., they require an appropriate definition for each optimization problem. For all other stages often default strategies are employed. The most popular strategies for this basic EA scheme are presented next from the point of view of structural optimization.

2.2.1 Initialization

The purpose of the initialization stage is to provide an initial population forming the basis of the subsequent genetic search. Typically, the individuals are randomly initialized, i.e., the design parameter values are completely chosen by chance, in order to sample the entire search space. Another possibility is to seed the initial population with prior knowledge by including some already known good solutions or by generating random values only in a specified range of the representation. Moreover, it might be useful to provide some known solutions which are only slightly modified, hence, the initial population contains a lot of prior knowledge. In case the random initialization leads to extremely non-competitive or even infeasible solutions not satisfying a given constraint, they can be excluded from the initial population.

The choice of the appropriate method is obviously problem-dependent. In particular for structural optimization problems where the evaluation takes a long time and only a relatively low total number of evaluations is possible a completely random initialization strategy might be inappropriate. In this case the optimization is started in a region of the search space close to the already known solution and the evolutionary process aims at finding fitter solutions nearby. Consequently, the evolutionary optimization loses a bit of its global approach, but the algorithm is still able to escape from local optima.

2.2.2 Mapping

In general, GAs require the implementation of a mapping process since they explicitly make a difference between the search (genotype) space and the solution (phenotype) space. The mapping stage can be simple, e.g. converting a binary string (genotype) to a real number (phenotype), but it can also be more complex. There is no definite answer to the question of whether this stage is really needed to improve the optimization performance or whether it is better to apply the variation operators directly in the phenotype space. However, several arguments support the opinion that the mapping process,

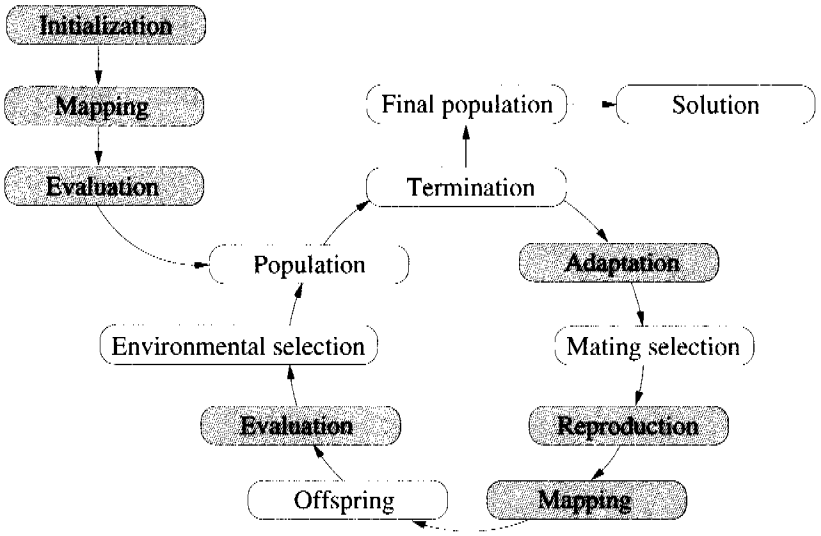


Figure 2.3: A basic EA scheme for structural optimization problems. Problem-dependent stages appear shaded.

known as *embryogeny* in biology, is important (cited from Bentley [81]).

- **Reduction of the search space.** Embryogeny permits highly compact genotypes to define the phenotypes. This reduction (often recursive, hierarchical, and multi-functional) results in genotypes with fewer parameters than their corresponding phenotypes.
- **Better enumeration of the search space.** Mapping permits two differently organized spaces to coexist. A search space designed to be easily searched can allow the EA to locate corresponding solutions within a hard-to-search solution space.
- **More flexibility to design smart recombination operators.** Maintaining a search space and a solution space allows for recombination operators changing the genotype itself. As an example, several genes can be grouped to entities or the genotype can be reordered during evolution.
- **Improved constraint handling.** Mapping processes can ensure that all phenotypes always satisfy all constraints, without reducing the effectiveness of the search process in any way, by mapping every genotype onto a legal phenotype.

The mapping strategy is the crucial point for evolutionary optimization of mechanical structures, since it defines the space of possible solutions. It is

extremely important that the representation and the corresponding mapping strategy as well as the variation operators are well-defined and match to each other. In Section 2.3 the representation issue is discussed in detail.

2.2.3 Evaluation

After the mapping process is completed, each new phenotype must be evaluated, i.e. the fitness value indicating the quality of the solution has to be assigned. In classical EAs the fitness is a scalar value, whereas for minimization problems a lower fitness value stands for better solutions and vice versa for maximization problems. This stage is normally the most expensive one, in particular for structural optimization problems where the evaluation typically incorporates numerical simulations taking minutes or even hours for a single evaluation. It is therefore of high importance to find a good compromise between simulation accuracy and evaluation time in order to perform an accurate and efficient optimization. Furthermore, the development of efficient optimization algorithms aiming at a reduction of the total number of evaluations also contributes to the optimization performance.

In general, fitness function formulations are required in order to assign a fitness value to a given solution. These fitness functions can have single or multiple objectives and often need to take constraints into account. Moreover, they can be of continuous or discontinuous, of smooth or noisy, and of static or continually changing nature. A large variety of different fitness function strategies exist which are most often tailored to the optimization problem. In Section 2.5 a static as well as an adaptive fitness function definition incorporating an arbitrary number of constraints are presented which are employed for all applications presented within this thesis.

2.2.4 Adaptation

After the initialization, the mapping, and the evaluation process are all performed once, the actual optimization cycle is entered with the adaptation stage. Only few of the known EA implementations run an adaptation stage. Their strategy parameters defining the operator rates, the fitness functions, the population size, etc. are altered during the optimization process. They are known as (self-)adaptive optimization algorithms which generally try to adapt the optimization strategy to the search state or even evolve optimal strategy parameters on the run. Eiben et al. [84] propose a comprehensive classification considering four aspects of adaptation:

1. *what* is changed: representation, fitness function, variation operators, operator rates, selection, population size, etc.,

2. *how* is it changed: deterministic, adaptive, or self-adaptive update mechanism,

3. on what *level* is it changed: population, individual, or gene level,

4. and based on which *evidence* the change is made: performance of operators, diversity of population, etc.

Item 2 deserves closer attention since these three different categories need some more explanation:

- *Deterministic* parameter control. No feedback from the search process is used and the strategy parameters are altered deterministically. Normally, the rule is applied periodically after a given number of generations.

- *Adaptive* parameter control. This adaptation strategy is based on feedback from the search process. The magnitude and the direction of change of the strategy parameters directly depend on predefined measures.

- *Self-adaptive* parameter control. The self-adaptation of strategy parameters is inspired by the idea of "evolution of evolution". The strategy parameters are directly encoded in the representation and are subject to mutation and recombination like ordinary genes.

Within this thesis the adaptation stage is used twofold. On the one hand, an adaptive operator-rate controlled EA (AORCEA) is presented in Chapter 4 aiming at minimizing the number of evaluations in order to increase the algorithm performance. On the other hand, an adaptive fitness function definition is introduced in Section 2.5 which is applied in chapters 5, 6, and 8. This adaptive fitness function aims at enforcing the compliance with some constraints if they are difficult to be fulfilled.

2.2.5 Mating selection

The selection of some solutions which are chosen to be parents of the subsequent generation is part of most of the EA implementations, although it is not essential for the evolution process. Other stages also exert evolutionary pressure, i.e., they prefer better solutions to propagate their genetic material to subsequent generations. Nevertheless, choosing the fitter solutions to be parents of the next generation is a common way of inducing selective pressure towards evolution of fitter individuals.

A variety of selection strategies exist, but typically one of the following is used:

- Fitness ranking. The current population is sorted according to the fitness values of the individuals and the probability of an individual to be selected as parent is based on its position in the ranking.

- Tournament selection. A given number of randomly picked individuals are compared with respect to their fitness values and the best one is selected as parent.
- Fitness proportionate selection. The probability to be selected as parent is based on the relative fitness scores of each individual, i.e., the fitter an individual the higher its chance to be selected.

This is a typical problem-independent stage not needing a customization to structural optimization problems.

2.2.6 Reproduction

Reproduction is the key stage of an evolutionary process, since it is responsible for the generation of offspring from the selected parent solutions. Two components required for evolution, namely variation and inheritance, are realized by the reproduction stage. It is crucial that the offspring inherit characteristics from their parents and that they are subject to variations. Therefore, two different types of variation operators, namely mutation and recombination, are typically implemented (except for the basic implementations of EP and ES where no recombination is used).

The purpose of mutation operators is to introduce variation into the optimization process by varying a single individual. This is necessary to prevent the search from getting trapped in local optima and to explore unknown search space regions with probably even better solutions to the problem. A lot of different mutation operators exist introducing more or less variation into the search process. The basic mutation operators used within this thesis are presented in Section 2.4.1.

Recombination operators, also known as crossover operators, require two or even more parent solutions. The genetic material of the parent solutions is partially exchanged leading to new offspring inheriting and recombining characteristics of their parents. Typically, for GA applications most of the offspring is generated by using such crossover operators since they are important for the convergence behavior of the optimization process, but for example EP and ES generally work without crossover operators. The implementation of these operators strongly depends on the chosen representation of the problem, thus only some basic operators are universally valid. The design of different basic recombination operators is discussed in Section 2.4.2.

2.2.7 Environmental selection

Once offspring is created from the selected parent solutions by applying variation operators, the environmental selection stage decides which of the new individuals will be part of the next generation. In most EA implementations

the population size is fixed, hence, for each individual which is inserted into the population an existing individual must be removed. A simple approach is to apply a generational strategy where all individuals in the population are replaced by the offspring. If additionally the *weak elitism* mechanism is activated, the fittest individual in the population cannot be replaced, thus, preventing the search process from losing the best solution. Other selection strategies are fitness-based as presented above, i.e., weak individuals in the population are replaced by fitter ones what is also known as steady-state strategy. Some other criteria leading to removal from the population can be the age of the solution, some constraint satisfaction measures or any other criteria introducing evolutionary pressure. For structural optimization problems, where the number of evaluations is a critical issue, steady-state strategies often perform more efficiently since a kind of implicit elitism is introduced.

2.2.8 Termination

A termination criterion is required in order to halt the optimization process. Typically, these criteria are based on the solution quality or on some time measures. The most simple quality-based termination criteria is to continue the algorithm until the fitness of the best individual reaches a given target fitness. Other criteria measure the convergence rates of the process or stop if for a given number of generations no improved solution could be found. In particular for optimizations which require computationally expensive fitness evaluations the primary termination criteria is the time or the available number of evaluations or generations, respectively.

The subsequent section discusses the representation issue in detail, since typically a mapping stage is employed for structural optimization problems.

2.3 Representation

Generally, the representation is basically an abstract description of an engineering design. In case of EC applications this representation is most often a genotype which is mapped to a phenotype in the mapping stage (cf. Section 2.2.2). Typically, the genotype is a multi-dimensional data container holding all the necessary information to describe the phenotype solution. The most straightforward genotype consists of a vector of genes, where each gene represents a dimension and corresponds to an attribute of the mechanical structure. As an example, the most simple genotype is made of a constant number of binary genes denoting the presence or absence of a feature. In more complex EC applications the genes are usually multi-valued.

The design of the representation is highly influenced by the optimization objective. In the case where novel design concepts or topologies are to be

found, the representation is called *open-ended* [15] indicating that a variable number of design entities is possible. On the other hand, the representation is called *parameterization*, if the topology is already defined in advance and only optimal shape and sizing are subject to optimization.

In general, representations are divided into *linear* and *non-linear*. The linear representations are organized as a vector of genes which can be of binary, integer-valued, or real-valued type. In contrast, the non-linear representations have more complex structures such as trees, bitarrays for FE-mesh based topology optimization [85], or mathematical graphs.

Furthermore, another distinguishing property of representations is their *length*. The number of genes of fixed-length genotypes remains constant during the entire optimization process, whereas the variable-length genotypes are allowed to change their length. Obviously, the variation operators need to be adapted in order to produce meaningful offspring from parents of possibly unequal length.

The definition of the representation and the definition of the mapping stage are obviously heavily interdependent. Thus, Gen and Cheng [86] discuss five major requirements characterizing good representations in terms of the mapping stage in engineering design problems.

Non-redundancy The mapping between the genotype space and the phenotype space $f : \mathbb{G} \rightarrow \mathbb{P}$ should be injective:

$$\forall x, y \in \mathbb{G} : x \neq y \Rightarrow f(x) \neq f(y), \tag{2.1}$$

i.e., two solutions which are different in the genotype space should also have varying attributes in the phenotype space. In practice this property is sometimes difficult to achieve. The worst case where a single genotype might have several solutions in the phenotype space, i.e.

$$\forall x, y \in \mathbb{G} : x = y \Rightarrow f(x) \neq f(y), \tag{2.2}$$

should be particularly avoided. Such properties can occur if the mapping procedure is based on stochastic or time-dependent rules.

Completeness Any solution in the phenotype space should have a corresponding genotype and is therefore accessible to genetic search. Thus, the mapping function should also be surjective:

$$f(\mathbb{G}) = \mathbb{P} \quad \text{or} \quad \forall x' \in \mathbb{P}\, \exists\, x \in \mathbb{G} : f(x) = x'. \tag{2.3}$$

In case the mapping function $f : \mathbb{G} \rightarrow \mathbb{P}$ is injective as well as surjective it has so-called bijective properties.

Legality With regard to the application of variation operators any mutation and recombination of genotypes should correspond to a solution. It is important to distinguish between *infeasibility* and *illegality*. Illegal means, that a particular genotype does not represent a phenotype for the given problem. On the other hand, an infeasible solution decoded from a genotype lies outside the feasible region violating some given constraints. As an example, in case of a structural optimization problem a maximum allowable stress value could be exceeded, what cannot be completely excluded by modifying the representation. Nevertheless, the space of infeasible phenotype solutions

$$\mathbb{P}_{infeas} = f(\mathbb{G}) - \mathbb{P}_{feas} \tag{2.4}$$

should be as small as possible, since the evaluation of these individuals uses computational resources. It is therefore of high importance to implicitly incorporate as much domain knowledge and constraints as possible. In particular manufacturing restrictions, e.g. minimum thicknesses of casting parts, or also design space limitations can easily be incorporated by defining a representation with design variable limits only allowing for feasible solutions.

Lamarckian property[1] The meaning of *alleles*, i.e. all possible values a gene can have, should not be context dependent. If the meaning of alleles for a gene is interpreted in a context-dependent manner, i.e., the meaning of a gene value depends on the other gene values as for example for representations of sorting problems like the Traveling Salesman Problem (TSP), the representation has the *non-Lamarckian* property. Then, the child solutions typically inherit nothing from their parents. Structural optimization problems typically have the Lamarckian property since each gene always codes the same attribute of the mechanical structure and its interpretation is independent from other gene values.

Causality / continuity Small variations in the genotype space should lead to small changes in the phenotype space. The appropriate definition of the mapping stage in combination with the variation operators is important for successful evolutionary search processes. The application of variation operators in the genotype space should preserve neighborhood in the corresponding phenotype structure. As an example, the causality requirement does not hold for binary coding because small changes of the binary string may lead to large changes in the real-valued parameter values. It is therefore favorable to use integer- or real-valued representations particularly in

[1]Note: this is different from the Lamarckian property known from numerous EAs (e.g. [87, 88]) which refers to whether an individual may be modified during the evaluation process, e.g. by "repair" mechanisms, or not.

structural optimization.

Next, the basics of a powerful representation concept developed for EC applications is briefly presented. It is an important component of all the representation concepts developed within this thesis and fulfills the above-mentioned requirements.

2.3.1 UniGene - the universal genotype

Typically, real-valued representations solve specific problems, e.g. the optimization of the shape of a given mechanical structure, but this concept is not suited to perform layout optimization as introduced above. If topology, shape, and sizing optimization should be combined, the representation needs to be able to code changes of all three categories. In particular discrete genes are required to realize changes of the topology. The most straightforward way is therefore to concatenate different representation types to a heterogeneous genotype. An excellent implementation of such a combined genotype is the *UniGene* concept introduced by König [80] and Wintermantel [89].

The universal genotype concept offers a collection of different gene types which can be arbitrarily combined to an entire genotype.

- *Float-gene* represents a real-valued design parameter with optional and arbitrary lower and upper limits.
- *Integer-gene* represents an integer-valued design parameter with optional and arbitrary lower and upper limits.
- *Bool-gene* represents a binary design parameter with state *true* or *false*.
- *Float-list-gene* represents a real-valued design parameter with arbitrary alleles.
- *Const-float-list-gene* represents a real-valued design parameter with equidistant alleles between a lower and an upper limit.
- *String-list-gene* represents a discrete design parameter of arbitrary values upon which no order or norm can be applied.

Typically, such an universal genotype is assembled from an arbitrary number of genes which are organized in a vector of constant length. For each gene type basic mutation and recombination functions are implemented defining how these operations change the value of the respective gene. The order of these genes is unchangeable, hence, recombination operators can be easily defined in a way that only genes of the same type mate with each other.

Furthermore, these gene types can also be integrated into any other fixed-length, variable-length, or non-linear representation as basic components.

The only restriction is that either the recombination operators have to preserve the order of the gene types or only genes of the same type are allowed to mate with each other in order to produce meaningful offspring solutions. Within this thesis the *UniGene* concept is a basis for all representation concepts presented in chapters 3 to 8 and it is therefore thoroughly explained in Appendix A. Due to the fact that different representation concepts are developed throughout this thesis, namely vector genotypes of constant and variable length as well as graph-based genotypes, they are discussed in the respective chapters.

2.4 Variation operators

The implementation of appropriate variation operators for the reproduction stage directly depends on the chosen representation. As an example, elementary GAs with binary string representation apply bit-flip mutation and one-point crossover, while basic ES with real-valued vectors use Gaussian mutation. Obviously, more complex representations, e.g. variable-length vectors or mathematical graphs, require customized variation operators preserving the structure of the representation. In other words, the variation operators have to comply with some rules ensuring that the representation itself is not inappropriately changed. As an example, the heterogeneous universal genotype presented in the previous Section 2.3.1 must not be manipulated with operators either changing the length or the size of the genotype.

A further requirement for structural optimization problems concerns the problem category introduced in Section 2.1. It is essential that the variation operators allow for changes of the topology, the shape, and the sizing, if all three categories should be subject to optimization, otherwise the optimization process cannot tap the full potential of structural improvements.

The main purpose of the variation operators is to further explore unexplored regions of the search space and to exploit already known domains. The implementation of the variation operators as well as their relative probability of application heavily influence the balance between exploration and exploitation. Thus, the determination and careful tuning of the operator rates is also an important issue with a possibly big impact on the optimization success. This problem is addressed in Chapter 4 where an adaptive operator-rate controlled EA (AORCEA) is introduced aiming at improving the optimization performance. The following subsections briefly explain the basic concepts of mutation and recombination operators for ordered and constant-length vector genotypes which are used in chapters 3 and 4. Furthermore, these variation operators are later taken as a basis for the development of the novel variation operators tailored to the representation concepts presented in chapters 5 to 8.

2.4.1 Mutation operators

The purpose of the mutation operators is to introduce variation into the evolutionary process. They usually act on a single individual by randomly changing some gene values. In general, four different mutation operators are used, namely *uniform mutation*, *deterministic uniform mutation*, *Gaussian mutation*, and *deterministic Gaussian mutation*. Therefore, each gene type needs to know how the respective mutation operator changes is current value.

In the case of uniform mutation every gene of the genotype is mutated with a relatively low probability. The new gene value is randomly computed within the admissible range, whereas the old gene value does not have any influence on the computation. A slight variation is the deterministic uniform mutation which alters a given number of genes of the genotype instead of using a probability for each of them.

The Gaussian mutation operator again changes the gene values of a genotype with a given probability. The current value v of a real-valued gene is altered according to the following formula

$$v' = v + \mathcal{N}(0, \sigma), \tag{2.5}$$

where σ is the standard deviation of a zero-mean normal distribution. Thus, depending on the value of σ this operator leads to relatively small variations of the old gene values. Analogously, similar Gaussian mutation mechanisms can also be defined for gene types which are not real-valued. Their implementations for each gene type can be found in Appendix A. Again, the deterministic Gaussian mutation is defined by the number of genes of an individual which are subject to modification.

2.4.2 Recombination operators

The recombination or also called crossover operators are responsible for the inheritance from parent properties to offspring solutions. In general, these operators are defined to generate two offspring solutions from two parents, but theoretically multiple parents may produce an arbitrary number of offspring as long as inheritance occurs. In general, two basic types of crossover operators can be distinguished:

Exchange crossover operators The standard procedure to create two offspring solutions from two parent solutions is to exchange single or multiple genes between the respective parents. This type of operator is independent from the data structure a single gene represents, but the genes to be exchanged must be at the same location in the respective genotypes. Typically, this prerequisite is fulfilled due to the fixed order and the constant length of the genotype.

Some examples for exchange crossovers are: *n-point crossover*, *uniform crossover*, and *intermediate crossover*. The classical n-point crossover chooses n sites separating the parent genotypes in $n+1$ segments before every second segment is exchanged. The uniform crossover simply exchanges a predefined or randomly chosen number of genes and the intermediate crossover uniformly swaps every gene with a given probability between the two parents.

Arithmetic crossover operators Arithmetic crossover operators create offspring by computing a kind of average value for every gene of the involved parents. These operators need more knowledge about the gene types they handle, i.e., rules defining the computation of the average values must be implemented for each gene type. Furthermore, these operators only act on single genes but not on groups of genes.

Two examples for arithmetic crossovers are the *hypercube crossover* and the *segment crossover*. Given two parent solutions $\mathbf{p^1} = [p_0^1, p_1^1, \ldots, p_n^1]$ and $\mathbf{p^2} = [p_0^2, p_1^2, \ldots, p_n^2]$, two offspring $\mathbf{o^1}$ and $\mathbf{o^2}$ are created according to the following rule

$$\begin{aligned} o_i^1 &= p_i^1 \cdot \gamma + p_i^2 \cdot (1-\gamma) \\ o_i^2 &= p_i^1 \cdot (1-\gamma) + p_i^2 \cdot \gamma \quad \text{for} \quad i = 0, \ldots, n, \end{aligned} \tag{2.6}$$

where γ is a random proportional factor and n is the length of the respective genotypes. For the hypercube crossover the factor γ is evaluated anew for each gene, whereas the segment crossover processes the parent genotypes with a constant factor γ.

In structural optimization the mutation and recombination operators are normally applied with a strong emphasis on recombination. A global crossover as well as a global mutation rate need to be defined and also a relative weighting specifying how often specific types of crossover and mutation are applied is required. The optimal configuration of global and relative weighting of operator rates cannot be analytically determined and is therefore based on lots of experience, adjusted by some rules, or adapted by the evolutionary algorithm itself.

2.5 Fitness evaluation

The fitness assignment is a part of the evaluation stage in any EC application. After the properties of an individual are evaluated, e.g. by a numerical simulation, they have to be rated in order to render the individuals comparable. An adequate fitness evaluation function assigning a unique and scalar fitness value to each individual is therefore a key issue in evolutionary optimization processes. Although the representation is generally constructed to

implicitly fulfill as many constraints of the optimization problem as possible, there are still constraints which cannot be influenced by the representation itself, e.g. maximum allowable mechanical stresses. Thus, the fitness evaluation should not only rate the design objective, but also a penalty mechanism is required punishing solutions conflicting with the constraints. A thorough overview of possible constraint handling strategies can be found in Yu [90].

2.5.1 Formulation of the fitness function

In contrast to multi-objective EAs (MOEAs) considering multiple objectives in order to find Pareto-optimal solutions, i.e. a set of alternative trade-offs, the optimization problems addressed within this thesis typically have a single objective, but they are also subject to an arbitrary number of constraints. The fitness formulation presented next is based on findings of König [80] and was implemented and further developed by the author.

The fitness function value $F(\mathbf{i})$ depending on the individual $\mathbf{i}$ is defined as a weighted sum

$$F(\mathbf{i}) = w_o D_o(\mathbf{i}) + \sum_j w_j D_j(\mathbf{i}), \tag{2.7}$$

where $D_o(\mathbf{i})$ represents the rating for the objective and $D_j(\mathbf{i})$ the rating for a specific constraint. Both w_o and w_j are the corresponding relative weights, respectively. This means that the objective and every constraint value must be mapped to an addend $D(\mathbf{i})$ of the fitness function, whereas $D(\mathbf{i})$ usually takes values in the range [0, 1]. Thus, none of the ratings becomes dominant compared to the others and can be easily scaled with the factor w in order to emphasize the importance of the objective or a constraint. Next, three different types of fitness functions for design objective, upper or lower, and target constraints are presented.

2.5.1.1 Design objective

An optimization problem with a design objective measure $O(\mathbf{i})$, e.g. the mass or the compliance of a structural part, can always be defined as a minimization problem

$$O'(\mathbf{i}) = \begin{cases} O(\mathbf{i}) & : \text{if } O(\mathbf{i}) \text{ to be minimized} \\ -O(\mathbf{i}) & : \text{if } O(\mathbf{i}) \text{ to be maximized,} \end{cases} \tag{2.8}$$

i.e., $O'(\mathbf{i})$ is to be minimized. The addend $D_o(O'(\mathbf{i}))$ of the fitness function should meet the following requirements:

- $D_o(O'(\mathbf{i})) \in [0, 1]$
- Relevant design improvements should be reflected in distinct decreases of $D_o(O'(\mathbf{i}))$.

- In the early search process the selection pressure should be strong and towards the end of the optimization it should continuously decrease. Note: This holds only if fitness proportionate selection is applied.

These requirements are incorporated into the following mapping function definition

$$D_o\left(O'(\mathbf{i})\right) = \left(a \cdot O'(\mathbf{i}) + b\right)^{\alpha}, \tag{2.9}$$

where the exponential factor $\alpha = 5$ is based on experience and a and b are scaling factors. The determination of a and b depends on two further values, namely O_{init} and O_{estim}. O_{init} represents an initial value of the design objective which results in a maximum fitness value of 1. On the other hand, O_{estim} is the estimated goal value which is supposedly achieved until the end of the optimization process. Its corresponding fitness value is set to 0.1, hence, for the mapping function it holds

$$\begin{aligned} D_o\left(O_{init}\right) &= 1 \\ D_o\left(O_{estim}\right) &= 0.1. \end{aligned} \tag{2.10}$$

As a consequence, the scaling factors a and b evaluate to

$$\begin{aligned} a &= \frac{1 - \sqrt[\alpha]{0.1}}{O_{init} - O_{estim}} \\ b &= 1 - a \cdot O_{init}. \end{aligned} \tag{2.11}$$

Figure 2.4 depicts the mapping function for a design objective with variable values for α, $O_{init} = 100$, and $O_{estim} = 0$.

2.5.1.2 Upper and lower limit constraint

For constraint values $C(\mathbf{i})$ which are not allowed to exceed or fall below a certain threshold value C_{limit}, the most straightforward penalizing approach would be a simple step function.

$$D_j\left(C(\mathbf{i})\right) = \begin{cases} 0 & : C\left(\mathbf{i}\right) \leq C_{limit} \\ 1 & : \text{else} \end{cases} \tag{2.12}$$

This penalty function exerts an immense selection pressure towards solutions fulfilling all the constraints. Unfortunately, such a definition is unable to rate the magnitude of constraint violation, hence, solutions only marginally violating the constraints would be excluded although they might hold valuable genetic information. As a consequence, a continuous constraint function is defined

$$D_j\left(C(\mathbf{i})\right) = \frac{1}{1 + e^{-\lambda\left(C(\mathbf{i}) - C_{limit} - \Delta\right)}} \tag{2.13}$$

giving constraint violating individuals a chance to remain part of the population. The two parameters Δ and λ are impractical to adjust the constraint

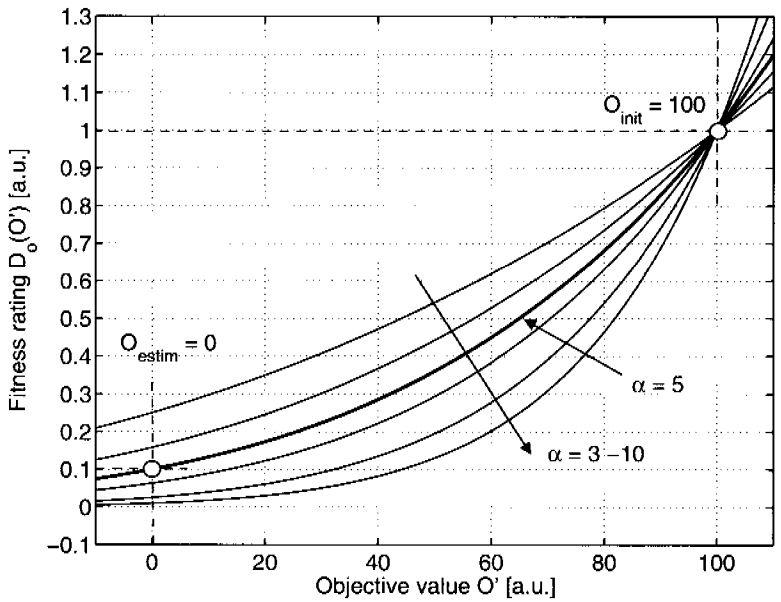


Figure 2.4: Fitness function for a design objective defined by O_{init} and O_{estim}.

function. Thus, another parameter, namely C_{feas_tol}, is introduced in order to contribute to the usability of the constraint function. According to the following definition

$$\begin{aligned} D_j\left(C_{limit}\right) &= D_{limit} = 0.01 \\ D_j\left(C_{limit} + C_{feas\_tol}\right) &= D_{feas} = 0.5, \end{aligned} \tag{2.14}$$

D_{limit} stands for the penalty value assigned to an individual exactly complying with the constraint. Moreover, D_{feas} corresponds to a constraint violating individual which is penalized with 0.5. The value C_{feas_tol} therefore defines the steepness of the constraint function. From Equations 2.13 and 2.14 the original parameters Δ and λ can be computed as

$$\begin{aligned} \lambda &= \frac{1}{C_{feas\_tol}} \left(\ln\left(\frac{1}{D_{limit}} - 1\right) - \ln\left(\frac{1}{D_{feas}} - 1\right)\right) \\ \Delta &= \frac{1}{\lambda} \ln\left(\frac{1}{D_{limit}} - 1\right) \end{aligned} \tag{2.15}$$

Figure 2.5 depicts a constraint function for a threshold value $C_{limit} = 5$ and different tolerance values $C_{feas\_tol} = 0.6, \ldots, 6$. The function formulation

can be used for lower ($C_{feas\_tol} \in \mathbb{R}^-$) and for upper limits ($C_{feas\_tol} \in \mathbb{R}^+$), respectively.

In particular for structural optimization problems this approach is extremely useful since optimized mechanical parts usually explore the limits of some constraints, e.g. the maximum compliance or also the maximum allowable stresses.

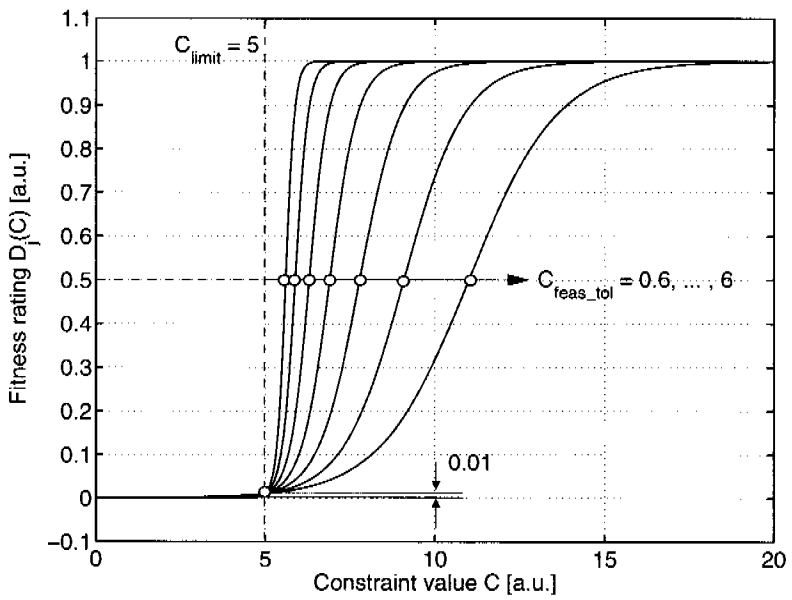


Figure 2.5: Penalty functions for an upper limit constraint defined by C_{limit} and C_{feas_tol}.

2.5.1.3 Target constraint

For some EC applications it is desirable to tune a certain property of the optimization model, e.g. the stiffness of a mechanical part, to a predefined target value C_{target}. A tolerance range C_{adm_tol} is additionally defined, since usually such target values cannot exactly be hit. Similar to the constraint function a smooth, but this time symmetric, and continuous function is defined as

$$D_j\left(C(\mathbf{i})\right) = \begin{cases} 0 & : |C(\mathbf{i}) - C_{target}| < C_{adm\_tol} \\ 1 - e^{-\frac{(|C(\mathbf{i}) - C_{target}| - C_{adm\_tol})^2}{2\kappa^2}} & : |C(\mathbf{i}) - C_{target}| \geq C_{adm\_tol}, \end{cases} \tag{2.16}$$

where $C(\mathbf{i})$ refers to the constraint value of individual $\mathbf{i}$ and κ defines the steepness of the function. Again, the value

$$D_j\left(C_{target} + C_{feas\_tol}\right) = D_{feas} = 0.5 \tag{2.17}$$

is defined in order to simplify the usage of the target constraint function. The parameter κ can therefore be determined as

$$\kappa^2 = \frac{\left(C_{feas\_tol} - C_{adm\_tol}\right)^2}{-2\ln\left(1 - D_{feas}\right)} \tag{2.18}$$

Figure 2.6 shows some example functions with a target value $C_{target} = 25$, a tolerance $C_{adm\_tol} = 2.5$ and variable values for $C_{feas\_tol} = 6, \ldots, 14$.

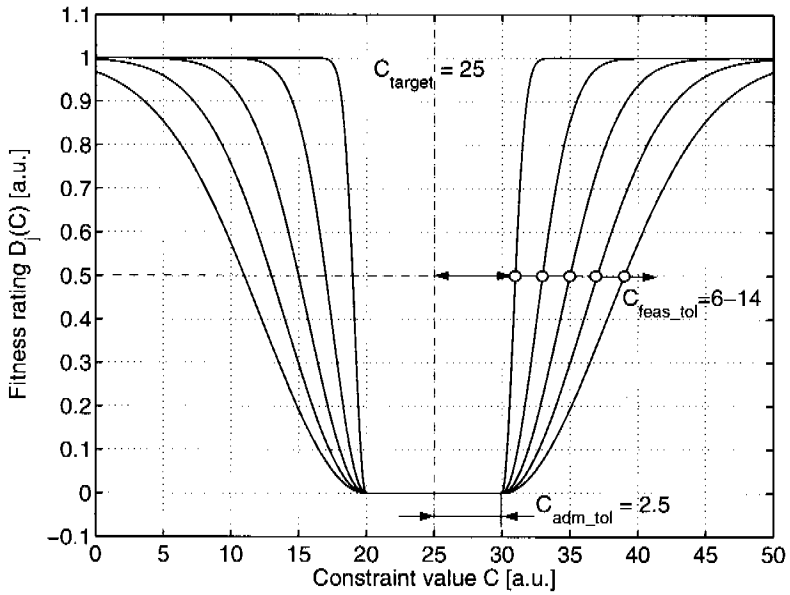


Figure 2.6: Penalty functions for a target constraint defined by C_{limit}, C_{adm_tol} and C_{feas_tol}.

2.5.2 Adaptive constraint functions

The above presented penalty function for upper and lower limit constraints (cf. Section 2.5.1.2) has proved its efficiency in several applications [80, 91, 92], but for some optimization problems the tuning of the parameters turned out to be impractical. In case the random initialization for a given problem almost exclusively produces constraint violating individuals, it may be extremely difficult to move the population towards the

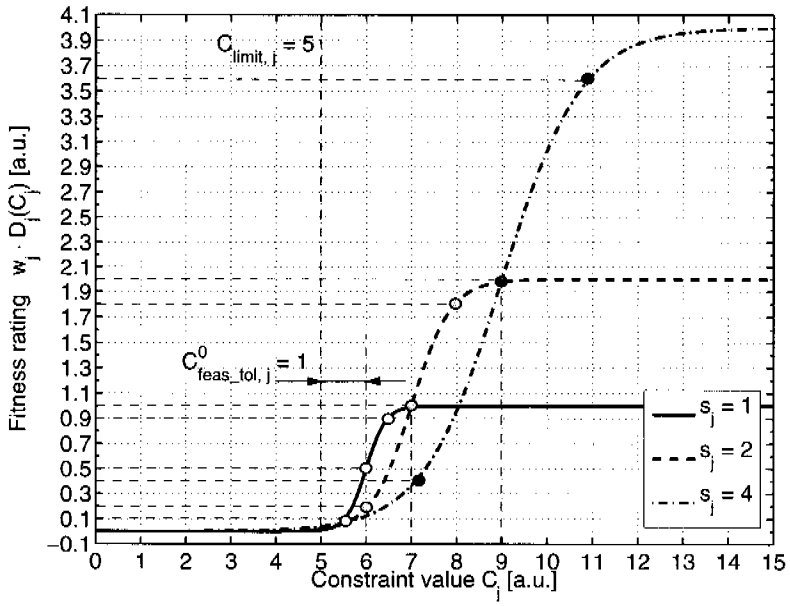


Figure 2.7: Adaptive penalty functions for the j^{th} limit constraint defined by $C_{limit,j}$ and $C^0_{feas\_tol,j}$.

constraint. This is because the initial definition of the constraint function must be chosen rather flat, otherwise all individuals are penalized with the same constraint rating value which is typically 1. Thus, in these cases it is common practice to start with an initial fitness function setup which is manually adjusted during the optimization and requires a restart of the entire process. Another phenomenon in conjunction with the constraint handling is that for some types of constraints the values can differ by orders of magnitude, e.g. margins of safety for some structural stability measures. Thus, it is almost impossible to find a solution which fulfills the respective constraint without manually adjusting the fitness function during the optimization process. A further development of the constraint functions is therefore the introduction of an adaptation strategy allowing for automatic redefinition of the fitness function during the optimization run.

The adaptation strategy needs to be based on a measure indicating the magnitude of constraint violation of the current population. First, average and median constraint values of the population were taken, but it turned out that these measures are inapplicable. For some constraint types these values can be almost random due to the large differences of the constraint values and scattering effects, respectively. Thus, the constraint value of the current

best individual $\mathbf{i}^*$ of the population $\mathbf{P}$ is taken as decision support in order to adapt the constraint function definition. The basic idea of the adaptation strategy is to force the compliance with the constraints by temporarily increasing their contribution to the overall fitness value.

The fitness adaptation Algorithm 2 is executed in each adaptation stage (cf. Section 2.2.4) of the EA. The current best solution $\mathbf{i}^*$ is checked for violation of each defined constraint. In case a constraint is not violated the constraint function remains unchanged, otherwise the adaptation algorithm is deployed. The constraint function is adapted, if the fitness rating $D_j(C_j(\mathbf{i}^*))$ of the best individual $\mathbf{i}^*$ for constraint j exceeds an upper rating limit D_{upper} or falls below D_{lower}. A scaling factor s_j with an initial value of 1 is multiplied by 2 in case the upper limit is exceeded and it is divided by 2 in the other case. The initial parameter $C^0_{feas\_tol,j}$ and the initial weight w^0_j are then multiplied by the scaling factor s_j, whereas the current value of s_j is kept for the adaptation stage of the next generation. In order to prevent extremely steep constraint functions, the scaling factor s_j cannot become smaller than its initial value.

The adaptation of the constraint function naturally involves a reevaluation of all individuals of the population, but since for each individual the constraint values are stored, the overall fitness values can be easily computed by re-rating the respective constraint values. Figure 2.7 depicts such an adaptive constraint function for an arbitrary constraint j with a limit

Algorithm 2 Adaptation of upper and lower limit constraint functions.

1: bool $adapted = false$
2: **for** $j = 0 <$ number of constraints **do**
3: **if** $D_j(C_j(\mathbf{i}^*)) > w_j \cdot D_{upper}$ **then**
4: $s_j = 2 \cdot s_j$
5: $w_j = s_j \cdot w^0_j$
6: $C_{feas\_tol,j} = s_j \cdot C^0_{feas\_tol,j}$
7: $adapted = true$
8: **else if** $D_j(C_j(\mathbf{i}^*)) < w_j \cdot D_{lower}$ and $w_j > w^0_j$ **then**
9: $s_j = 0.5 \cdot s_j$
10: $w_j = s_j \cdot w^0_j$
11: $C_{feas\_tol,j} = s_j \cdot C^0_{feas\_tol,j}$
12: $adapted = true$
13: **end if**
14: j = j + 1
15: **end for**
16: **if** $adapted = true$ **then**
17: Reevaluate fitness of population $\mathbf{P}$
18: **end if**

value of $C_{limit,j} = 5$ and an upper rating limit of $D_{upper} = 0.9$ and a lower rating limit $D_{lower} = 0.1$. The rating limit values are chosen based on experience since they proved to be efficient for a variety of test problems.

Next, some brief notes on the implementation of EC systems within this thesis are presented.

2.6 Implementation - the Evolving Objects library

All the EAs used within this thesis are implemented based on the Evolving Objects (EOlib) [93] library, an object-oriented framework for EC applications. EOlib provides a set of C++ classes for the evolution of arbitrary data structures (objects) which fulfill some simple preconditions, namely initializability, selectability, replicability, mutability, and combinability [94]. The algorithms programmed with EOlib are not limited to the basic EA branches introduced in Section 1.1.3.1, but arbitrary combinations and new developments in each EA stage, e.g. reproduction or evaluation, can be easily integrated. This generic concept makes the EOlib framework superior to the few other available libraries such as Matthew's GAlib [95] or PGAPack [96].

This chapter is concluded with a survey of representation and adaptation concepts presented within this thesis.

2.7 Survey of representation and adaptation concepts

Within this thesis several different combinations of representations, variation operators, and adaptation strategies are applied. It is important to emphasize that the representation concept and the corresponding variation operators are highly dependent in any EC application. As a consequence, the following chapters thoroughly present the representation as well as the variation operators and, where employed, the adaptation strategies. An overview is given in Table 2.1.

Application	Chapter	Representation	Adaptation
CFRP rim	3	vector genotype	–
AORCEA	4	vector genotype	operator rates
Variable-length genotype	5	variable-length vector genotype	fitness
coGraphTruss	6	graph genotype	fitness
eoGraphLaminate	7	graph genotype	–
eoGraphCAD	8	graph genotype	fitness

Table 2.1: Survey of representation and adaptation concepts.

Chapter 3

Application: CFRP racing motorcycle rims

This chapter is based on the publication: *Development of CFRP racing motorcycle rims using a heuristic evolutionary algorithm approach* [91].

3.1 Introduction

In motorcycle racing it is absolutely crucial to permanently pursue improvements and new developments in order to remain competitive. A variety of components contribute to an outstanding racing performance, e.g., engine performance, aerodynamic properties, wheel grip, the drivers' skills, but in particular the motorcycle rims significantly influence the overall performance.

The quality of a motorcycle rim can directly be quantified through its mechanical properties such as stiffness and strength. The stiffness is particularly important in turns of high speed where vertical and lateral loads resulting from the motorcycle's weight and centripetal effects introduce bending loads into the rear and the front rim. Furthermore, the motorcycle rims must not fail when maximum loading occurs. The front rim is heavily loaded when full braking is applied. The maximum loading of the rear rim occurs at maximum acceleration of the motorcycle once the front rim loses ground contact.

In addition to the mechanical properties the mass and the moments of inertia of a motorcycle rim should be as low as possible. On the one hand, the rim mass contributes to the overall mass of the motorcycle and it belongs to the so-called unsprung mass crucially influencing the handling of the motorcycle on uneven race tracks. On the other hand, low moments of inertia of the rims are much more important. As a matter of simple physical principles, a lower moment of inertia with respect to the rotation axis of the rim allows faster acceleration and deceleration of the wheels

and therefore of the racing motorcycle. Moreover, changing the direction of the motorcycle creates gyroscopic forces whose magnitudes are directly correlated to the moments of inertia of the wheels. It is therefore crucial to reduce the moments of inertia - in particular of the front rim - in order to improve the handling, the performance, and therefore the competitiveness of the racing motorcycle.

Nowadays, magnesium alloy rims that are even lighter than aluminum rims are state-of-the-art in motorcycle racing. These magnesium alloy rims are technologically sophisticated and do therefore not offer a great potential of further mass reduction. Thus, the newly developed rims are manufactured from carbon fiber reinforced plastics (CFRP) allowing for a significant reduction of mass and moments of inertia due to superior stiffness and strength properties of the material.

The rims are manufactured by combining unidirectional and woven CFRP plies, whereas the mechanical properties and the orientation of each ply directly influence the mechanical properties of the rim structures. Consequently, the highly complex issue of determining the optimum stacking sequence of the plies needs to be addressed. Only the application of advanced optimization methods allows for the consequent exploration of the mass reduction limits. Obviously, the integers representing the quantity of plies and the discrete parameters determining the orientation of the plies render the search space of the optimization task extremely discontinuous. Furthermore, a variety of constraints have to be considered during the optimization in order to produce feasible design solutions. Consequently, the optimization problem cannot be efficiently solved by using gradient-based optimization techniques so that evolutionary optimization methods need to be chosen.

Section 3.2 gives a brief overview of the required steps to apply EAs to the rim optimization problem. The set-up of the necessary simulation models is outlined in Section 3.3, whereas the representation is detailed in Section 3.4 and the optimization process is the subject of Section 3.5. Within the scope of this project a front and a rear rim are developed. The methodology of the optimization process is demonstrated by means of the front rim but the optimization results are presented for both motorcycle rims in Section 3.6. Finally, a short conclusion and an outlook are given in Section 3.7.

3.2 Evolutionary algorithms applied to the rim optimization problem

Before Evolutionary Algorithms can be used for the optimization of any structural part, a preparatory step has to be carried out. A simulation model of the structure to be optimized is required in order to evaluate the fitness value for each individual. A CAD-model of the motorcycle rim is

prepared using the CAD-software CATIA V5 defining the geometric shape of the structure (cf. Section 3.3.1). In this project the geometric shape of the motorcycle rim remains unchanged during the optimization process, only the stacking sequence of the composite laminate is altered. The CAD-model forms the basis of a FE-model generated in ANSYS used to evaluate stiffness, strength, and mass properties of the motorcycle rim (cf. Section 3.3.2).

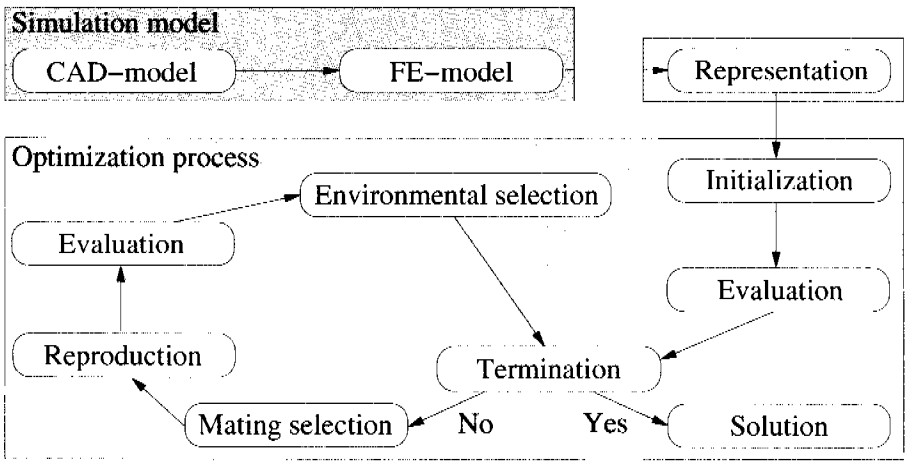


Figure 3.1: Working schedule of EAs applied to structural optimization problems. The definition of the representation is a preliminary step of the optimization process.

Figure 3.1 illustrates a general working schedule for structural optimization problems addressed with an evolutionary approach. After defining the simulation model, the optimization process can be set up. The representation is the key to an efficient and successful optimization. It has to be defined which parameters, e.g. the number of plies at a given position or the orientation of a certain ply, are modified during the optimization process. Obviously, there are thousands of possibilities to represent laminated structures. The structure can be subdivided into an arbitrary number of sections having different stacking sequences and requiring therefore a lot of optimization parameters defining the number of fiber plies and their orientation. The number of optimization parameters defines the size of the genotype and accordingly the size of the search space. A large number of parameters expands the search space and generally requires a greater number of evaluations to explore the search space sufficiently and to finally converge to an optimum solution. Therefore, it is extremely important to choose an appropriate representation rendering possible reasonable optimization results.

The last step before starting the optimization process is the definition

of the evaluation step. A fitness function (cf. Section 3.5.1) mapping the evaluated stiffness, strength, and mass measures from the FE-simulation to a unique fitness value needs to be defined.

After these preparatory steps the actual optimization process can be started. A starting population is initialized based on random initialization. Afterwards, the optimization loop containing evaluation, mating selection, crossover, mutation, and selection is iteratively run until a given stopping criterion is reached and an optimum design solution is found.

In the following sections all the above mentioned steps from the simulation model to the optimization process are discussed in detail.

3.3 Simulation model

For both the front and the rear rim a FE-simulation model has to be set up. Since this project is concerned with a complete new development of the motorcycle rims, CAD-models only defining the geometric shapes, but no material or other structural properties, are created first. Then FE-models defining the stacking sequences are derived from these CAD-models. Finally, the load cases for the simulation of the mechanical behavior are defined.

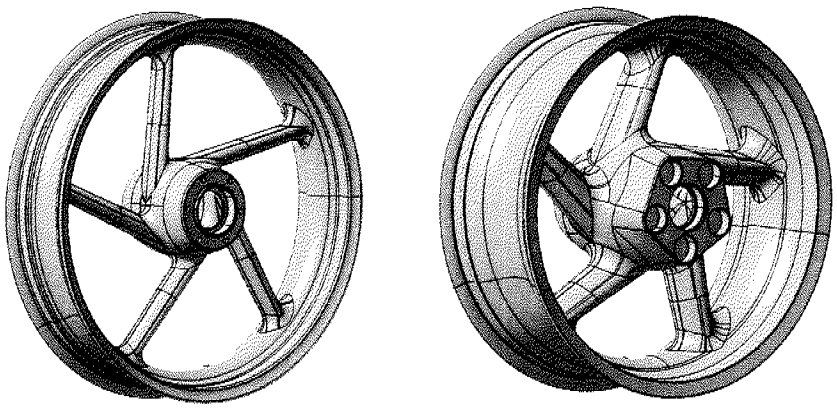
3.3.1 CAD-model

The CAD-models define the geometric shapes of the motorcycle rims and are therefore heavily constrained to meet a variety of requirements. There are some interfaces to adjacent parts that have to comply with small tolerances in order to guarantee faultless function. Probably the most important interface is situated between the tires and the rims since there all acceleration and deceleration forces are transmitted through friction forces. The geometric shapes of the rim beds are therefore given by the geometry of the tires. Furthermore, the front rim has two more interfaces to the brake disks transmitting the braking forces over the tires to the race track. The rear rim has only one interface to a brake disk, but additionally the drive torque is transmitted from the engine via the chain through five rubber inserts to the rear rim. Finally, the rims have bearing carriers forming the connection to the motorcycle suspensions.

Only the spokes and adjacent regions of the hub remain to the designer's creativity, whereas still a constraining factor has to be considered. The width of the spokes is limited by the design space boundaries given through the brake caliper.

It is decided to use a five-spokes design particularly due to the customer acceptance, but also stiffness properties, load transmission qualities, etc. are not less important. The bending stresses introduced by maximum drive or brake torque are reduced by connecting the spokes tangentially to the hub. The design of the rims and in particular of the spokes, see Figure 3.2,

are defined without running shape optimizations, since at this stage the optimization software tools have not allowed for a combined optimization of geometric shape and stacking sequence of the laminates.



(a) Front rim (b) Rear rim

Figure 3.2: CAD-models of the rims.

3.3.2 FE-model

The FE-models of the front and rear rims are set up in ANSYS by importing surface models from the CAD-software CATIA V5. The surface model is then meshed with layered 4-node isoparametric shell elements (SHELL181) admitting the definition of laminate properties, i.e. the stacking sequence consisting of single CFRP plies.

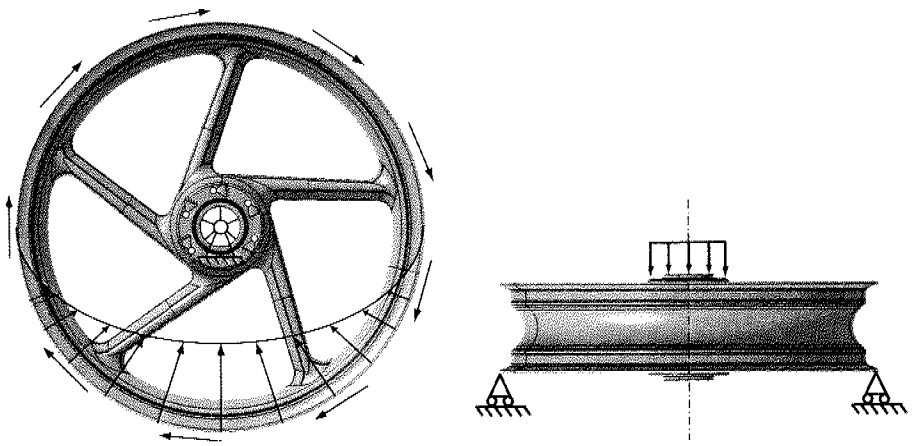
The purpose of the FE-model is to simulate the critical load cases occurring during a race and to prove the mechanical strength of the rims. Furthermore, the FE-analysis is also applied for the estimation of the stiffness properties of the rims. The rims must not be too compliant even if they would not fail due to mechanical stresses. Consequently, two load cases per front and rear rim have to be defined, one simulating the maximum load case and another providing a stiffness measure. For brevity only the FE-model of the front rim is discussed. Both load cases for the front rim are illustrated in Figure 3.3.

Maximum loading of the front rim occurs in full braking stages when the rear wheel loses ground contact. All the braking forces are transmitted through the front rim to the race track causing a torsional loading of the structure. The brake torque is assumed to be equally distributed along the circumference of the rim bed which is in direct contact with the tire. At the bearing carrier the rim is axially and laterally supported and the holes

for the bolted joints to the brake disk carriers are supported to block the rotational degree of freedom.

Additionally, the ground reaction force has to be superposed after it is multiplied with a scaling factor in order to take vertical impacts into account. The ground reaction force is assumed to follow a cosine distribution along the lower half of the rim circumference. In a former project [97] this assumption produced the most accurate simulation results compared to practical deformation tests. For this load case lateral forces introduced by centripetal effects can be neglected.

For the estimation of the stiffness properties a lateral load case is evaluated. The rim is laterally supported at one outer rim bed and a lateral force is applied on the hub. The deflection value is compared to results from tests with state-of-the-art magnesium alloy rims. The final aim is to approximately match the stiffness properties of these.



(a) Maximum brake torque and ground reaction

(b) Lateral load case

Figure 3.3: FE-model load cases of the front rim.

3.4 Representation

Based on the FE-model of the front rim the representation is set up by defining the optimization parameters of the rim model. The optimization parameters span the search space and simultaneously determine the appearance of the model in the solution space. Since the geometric shape of the motorcycle rim is fixed, no shape parameters appear. Only parameters defining the stacking sequence of the composite laminate are admissible.

3.4.1 Optimization model

Obviously, the stacking sequence should not be identical for the entire rim model since there are many differently stressed parts of the structure. It is important to only utilize as much fiber material as absolutely necessary in each section of the rim in order to tap the full potential of mass and moment of inertia reduction. Consequently, the rim structure has to be subdivided into several sections having different stacking sequences. The boundaries of these sections are defined in consideration of the force flow in the structure enabling the optimization to locally reinforce highly stressed regions by applying additional plies. The number of sections directly influences the number of optimization parameters because each section with a different stacking sequence requires additional optimization parameters. In consideration of the rule of thumb that the number of parameters to be optimized should not exceed the population size to get reasonable optimization performance beyond pure stochastic search, the number of different sections cannot be arbitrarily high. In consideration of the available computing power a maximum of approximately sixty to eighty optimization parameters has been chosen.

It is decided to subdivide the rim structure into four major domains having different stacking sequences of the composite laminates. The first domain includes the hub and the spokes, the second domain incorporates the rim bed, and the third domain defines the region where the first two domains overlap each other. Domain number four has the same base laminate as the first domain, but some additional unidirectional plies are added in order to locally reinforce this highly stressed region, i.e., the transition zones between the hub and the spokes. Figure 3.4 illustrates the subdivision into these four domains with the respective optimization parameters.

For the first two domains base composite laminates are defined consisting of a maximum number of ten plies each. The choice of the maximum number of ten plies is based on an antecedent FE-simulation with woven laminates only leading to a feasible design solution. Both base composite laminates require thirty parameters in order to define their stacking sequences, respectively. Ten parameters determine whether each of the ten plies is active or not, ten parameters define the material type, and ten further parameters define the ply orientations. In the third domain both base laminates are partially overlapped in order to guarantee a proper connection between the first two domains, hence, no additional parameters are required. For the fourth domain an integer parameter is added to the genotype to define the additional number of unidirectional reinforcement plies.

The optimization model finally consists of 61 parameters to be optimized in order to achieve the optimization objective, i.e. minimization of the structural mass, in consideration of stiffness and strength constraints. They are organized in a vector genotype and all

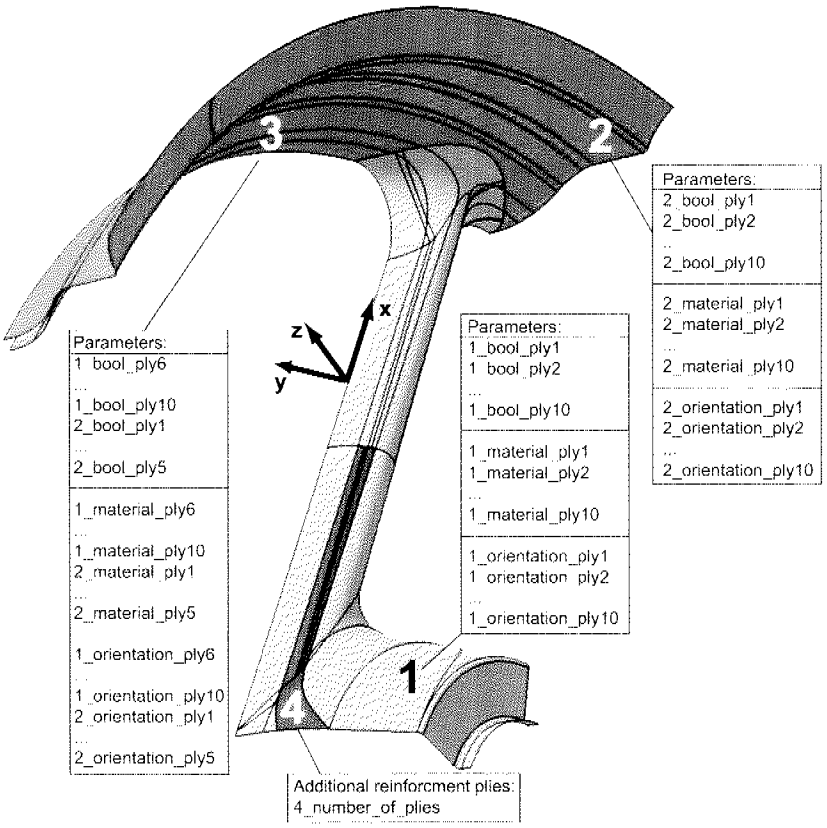


Figure 3.4: Schematic illustration of the four major domains with the corresponding optimization parameters.

3.4.2 Implementation of the representation

The choice of the optimization parameters determines how the stacking sequence of the rim can be changed during the optimization process. Basically, there are three different parameters describing the stacking sequence of a structural part made from CFRP prepregs, i.e. carbon fibers impregnated with thermosetting epoxy resins.

- The *number* of fiber plies has to be determined.
- A variety of prepreg *materials* are available with different fiber layouts. In this project an unidirectional carbon laminate and woven carbon laminates are used, whereas the woven fabric is available in two different thicknesses.

- The *orientation* of each orthotropic ply needs to be defined.

The representation of these properties is addressed by using the universal genotype concept introduced in Section 2.3.1 and further detailed in Appendix A. For each ply of the composite laminate a boolean, a string-list, and a const-float-list gene need to be defined.

- A maximum number of plies is defined for the stacking sequence in each section of the rim. The boolean gene determines whether a ply of the stacking sequence is active (*true*) or not (*false*). By setting the boolean value to *false* for some plies they can be omitted to reduce the rim mass. This procedure is necessary because the evolutionary operators always require constant-length genotypes. It is therefore impossible to only represent all existing plies and removing the genes defining the plies that are eliminated from the genotype.

- The string-list gene identifies the material of a ply. For the manufacturing of the motorcycle rim three different prepreg types are selected. The first prepreg type is a standard unidirectional carbon laminate with a thickness of 0.15mm represented by the string *t_ud*. The second and third prepreg types are woven carbon laminates (orientation 0° and 90°) having thicknesses of 0.25mm and 0.5mm, respectively. They are represented by the strings *t_300* and *t_1100* according to their product names[1]. All these materials are defined in the FE-model and can therefore arbitrarily be chosen by the optimization algorithm.

- The third gene type, the so-called const-float-list gene, determines the orientation of the chosen prepreg for each ply. Due to the limited manufacturing accuracy of approximately 5° it is useless to admit arbitrary orientation angles, because the search space would then be unnecessarily large. Thus, only a few possible values (alleles) for the ply orientation are taken into account. These values need to span the whole range from -90° to 90° in order to tap the full potential of the CFRP materials. This range is subdivided into twelve equidistant subranges leading to possible ply orientation values of -90°, -75°, -60°, to 60°, 75°, and 90°. In fact, the range for the woven laminates only needs to span a range from 0° to 90°, but the representation does not allow to distinguish between two different ranges for unidirectional and woven laminates since the laminate type is chosen independently from the ply orientation. The increment of 15° is larger than the theoretically possible manufacturing accuracy, but it is considered to be sufficient small to not lose a potentially best solution.

In addition, an integer gene is needed for the fourth domain to locally define the number of additional unidirectional reinforcement plies with a 0°

[1] LTM26-EL component prepregs. http://www.acg.co.uk

orientation. Figure 3.5 illustrates schematically the parameterization of an arbitrary section.

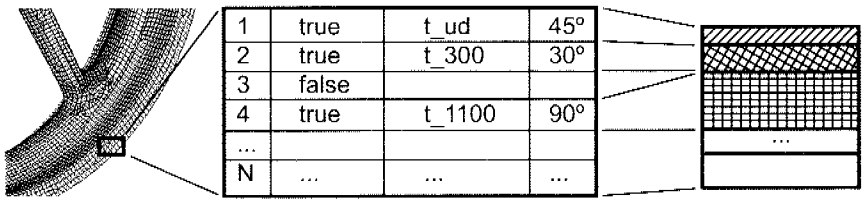


Figure 3.5: Schematic illustration of the representation concept.

3.5 Optimization process

The variation operators presented in Section 2.4 are applied during the optimization process. A fitness proportionate mating selection scheme and a steady-state replacement strategy introducing environmental pressure drive the evolutionary process.

For the purpose of rating the individuals, a fitness function needs to be defined combining the optimization objective with the optimization constraints. These fitness formulations for the rating of the evaluated rim individuals are presented in Section 3.5.1 and in Section 3.5.2 the software tool DynOPS controlling the optimization loop is briefly explained.

3.5.1 Fitness function

The optimization objective is to minimize the front rim's mass and accordingly its moment of inertia. Simultaneously, two constraints have to be fulfilled, i.e. the rim must not collapse under maximum loading and it has to provide a target stiffness for sufficient handling properties.

The strength criterion is based on the well known Tsai-Wu criterion [98] for laminated structures. The Tsai-Wu index must remain below the limit value of 1 in order to prevent damage of the structure. The stiffness criterion relies on practical stiffness tests with magnesium alloy rims, whereas the CFRP rim should approximately achieve the same stiffness properties for the lateral load case as outlined in Section 3.3.2. The lower deflection limit is set to 0.13mm and the upper limit is set to 0.23mm. Thus, the optimization can be formulated as follows

The objective and both constraints are mapped to a single fitness value according to the fitness definition presented in Section 2.5.

Minimize	mass				
subject to:			Tsai-Wu index	$<$	1
	0.13mm	$<$	Deflection value	$<$	0.23mm

The mapping function for the optimization objective, i.e. the mass, is defined by $O_{init} = 1500$ grams and $O_{estim} = 900$ grams. Figure 3.6 illustrates this mapping function. Since the FE-model is simplified compared to the CAD-model the evaluated mass of each individual is below the effective mass of the corresponding manufactured rim. This difference does not affect the optimization itself, because the fitness portion of the mass is a relative measure for the quality of the rim.

The mapping function of the upper limit constraint, i.e. the Tsai-Wu index, takes two parameter values, namely $C_{limit} = 1$ and $C_{feas\_tol} = 0.1$. Figure 3.7 illustrates the respective mapping function used for the optimization of the front rim. The second constraint is formulated as a target constraint and the corresponding mapping function is depicted in Figure 3.8 with the defining parameters $C_{limit} = 0.18$mm, $C_{feas_tol} = 0.12$mm, and $C_{adm_tol} = 0.05$mm.

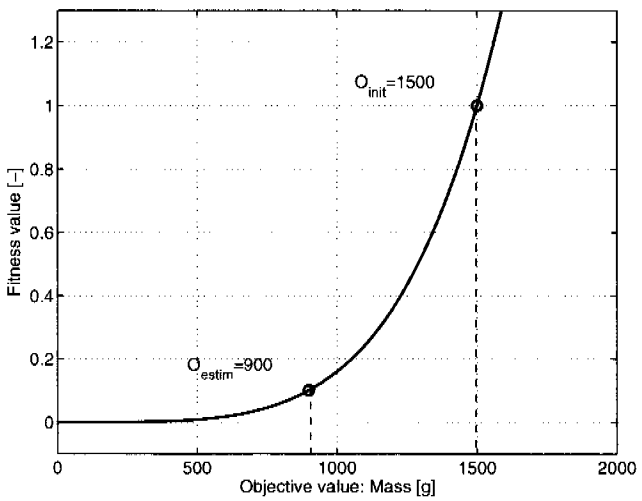


Figure 3.6: Mapping function for the optimization objective (mass).

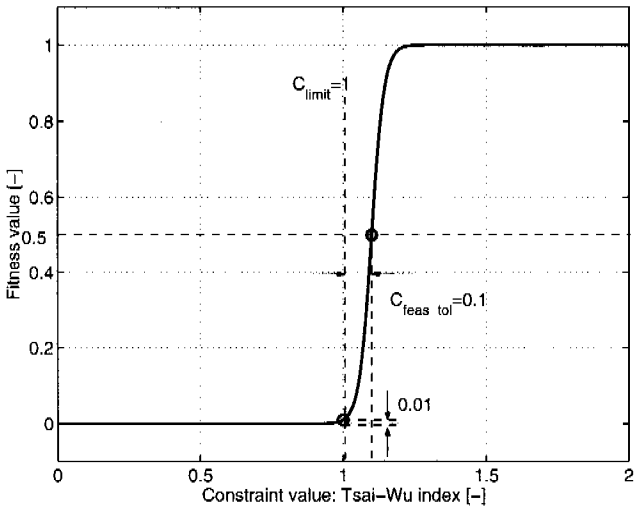


Figure 3.7: Mapping function for the upper limit constraint (Tsai-Wu index).

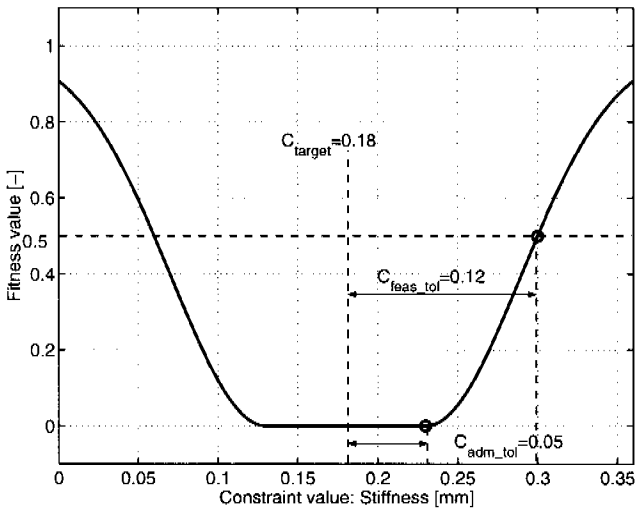


Figure 3.8: Mapping function for the target constraint (stiffness).

3.5.2 DynOPS controlled optimization loop

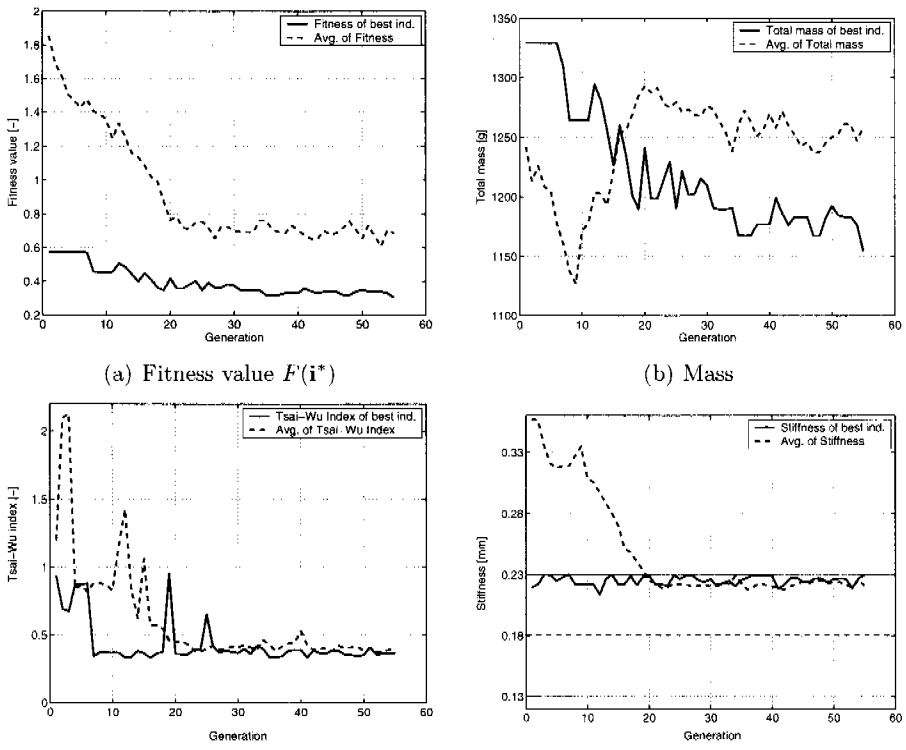
The execution of the optimization process is based on a variety of components. The Evolving Objects (EO) library, the representation based on the universal genotype concept, the evaluation of the FE-model, and the assignment of a fitness value to the evaluated individuals need to be managed. A proprietary developed software tool called DynOPS (**Dyn**amic **O**ptimization **P**arameter **S**ubstitution) written in C++ is able to cope with this complex task. DynOPS connects the EO library with any simulation software that can be started in batch-mode, e.g. ANSYS, CATIA V5, etc. The evaluation of the individuals during the optimization loop is most often computationally expensive. For this reason DynOPS runs the evaluation tasks in parallel by distributing them to a potentially heterogeneous hardware environment. DynOPS consists of four different modules:

- **Optimization engine** The Evolving Objects (EO) library provides the necessary framework to apply EAs in general. In connection with the universal genotype, an optimization engine for structural optimization problems is established.
- **Text file interface** DynOPS creates text-based input files to run arbitrary simulation software (in this project ANSYS for the evaluation of all individuals) in their batch mode. A prototype text file has to be provided with the specification of the optimization parameter positions within this file. DynOPS then generates new input files based on the new parameter values provided by the optimization engine that are sent to the simulation software for the evaluation of the respective individual. After these individuals are evaluated DynOPS reads the result values, i.e. objective and constraint values, from output text files created by the simulation software in order to evaluate the overall fitness value according to the defined fitness function.
- **Process manager** The process manager specifies a sequence of programs with their adequate input files to be executed during the optimization process, whereas the input and also the output files can be of any data format. It is possible to use files of arbitrary format, e.g. a modified CAD geometry that is passed to a Finite Element analysis.
- **Parallelization** DynOPS is able to evaluate the individuals of a population in parallel using the PVM (**P**arallel **V**irtual **M**achine) library [99]. The parallel machine is set up based on the user-specified list of processors that need to be connected, whereas DynOPS manages the evaluation tasks in a queue and distributes them to available processors.

For more detailed information on DynOPS see König [80].

3.6 Optimization results

Figure 3.9 illustrates some convergence plots of the front rim optimization. In each plot the objective and constraint values of the best individual having the lowest overall fitness value $F(\mathbf{i}^*)$ within the population of a generation as well as the average value of the entire population are presented. The plots show the first 55 evaluated generations; further evaluations have not found significantly improved design solutions and are therefore omitted from the plots. The optimization run is performed with 50 individuals per generation since the computing time is also a kind of an optimization constraint. This number is slightly below the rough rule of thumb saying that the number of individuals per generation should be at least the number of optimization parameters (61). Instead, the probability of mutation is chosen rather high in order to ensure a sufficient exploration of the entire search space.



(a) Fitness value $F(\mathbf{i}^*)$

(b) Mass

(c) Tsai-Wu index

(d) Stiffness value

Figure 3.9: Convergence plots of the front rim optimization.

Regarding the convergence plots it is important to note that from the

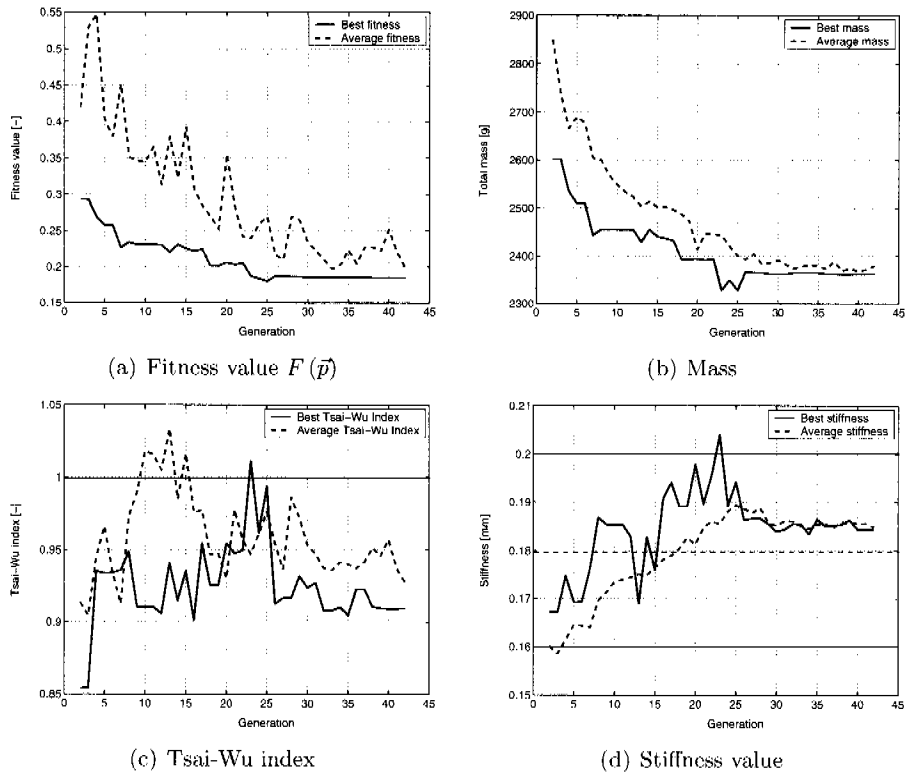
beginning of the optimization process the respective best solutions always fulfill both the strength and the stiffness constraint. The overall fitness value is reduced to approximately 50% of the initial fitness value, whereas the mass of the best individual is reduced by about 15%. The average of the fitness value is reduced simultaneously indicating a decreasing diversity in the populations. Obviously, these results are heavily influenced by the chosen representation. If the number of independent domains would be chosen much higher, e.g. ten instead of four, there would be an even greater potential of mass reduction. The stacking sequences could be adjusted more precisely to the loading in each domain, but then the optimization would be computationally more expensive since this implicates a larger search space. Furthermore, it is extremely important to establish a representation leading to feasible designs from the beginning of the optimization, otherwise the optimization might be incapable of finding any feasible design solution at all.

Finally, only the stiffness constraint restricts the optimization since the optimum rim is manufactured from a lot of thick woven layers to achieve the stiffness target value. The optimization of the front rim clearly shows that it is the stiffness constraint that prevents the rim from becoming lighter. Under the given load cases the front rim is not at risk to fail due to high stresses what is indicated by a Tsai-Wu index of approximately 0.4.

The results of the rear rim optimization are presented in Figure 3.10. Basically, the results are similar to the optimization results of the front rim optimization. The geometric shape of the rear rim is different from the front rim shape and therefore another representation and slightly different mapping functions are defined. The optimization is also run with 50 individuals per generation and is stopped after 42 generations. The fitness value of the best individual in each generation is reduced by approximately 38%, whereas the mass is reduced by about 9%. The rear rim optimization converges to a solution where the strength and the stiffness constraint are quite close to their allowable limits.

Both optimizations finally lead to manufacturable stacking sequences of the laminates consisting of all three given materials. The ply orientation angles span the range from -90° to 90° whereas some directions are particularly reinforced according to the force flow in the respective sections. The maximum number of plies is not utilized in all sections of the rim, i.e. the mass of the rims is reduced by choosing small prepreg materials and by eliminating entire plies from the stacking sequence.

The optimization results are the basis for the realization of both motorcycle rims which are shown in Figure 3.11. Details of the manufacturing process are not discussed within the scope of this thesis. Finally, when comparing the mass of the novel rear rim (2220 grams) with state-of-the-art magnesium alloy rims (approx. 3000 grams) a noticeable advantage of about 800 grams or 25% results. For the front rim the mass is 1650 grams what is



(a) Fitness value $F(\vec{p})$

(b) Mass

(c) Tsai-Wu index

(d) Stiffness value

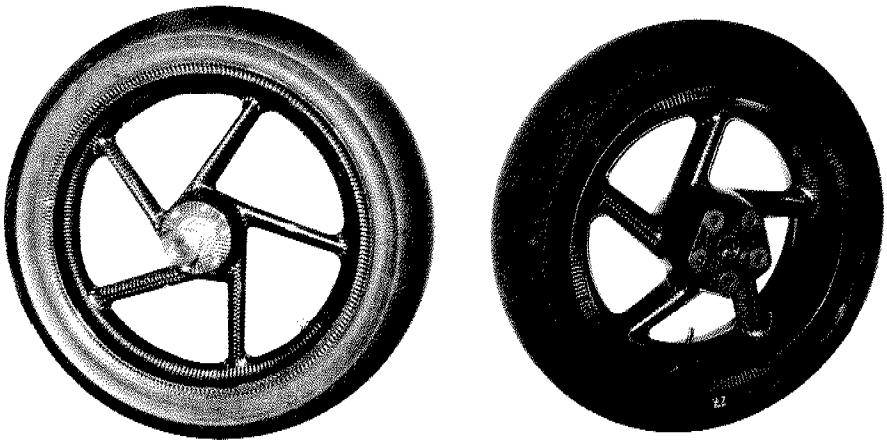
Figure 3.10: Convergence plots of the rear rim optimization.

a reduction of 15% compared to the magnesium alloy rims (1950 grams).

3.7 Conclusion

EAs are a well-suited tool for the optimization of laminated structures. The presented optimization problem demonstrates their applicability particularly to problems with discrete search spaces. Only the application of the heterogeneous genotype and an adequate representation technique make this optimization possible and successful. The formulation of the fitness function consisting of smooth constraint mapping functions steers the optimization into an appropriate direction, finally leading to significantly improved CFRP motorcycle rims that hardly could be designed by human intuition.

A major drawback of the evolutionary optimization at hand is that it requires equal-length genotypes because of the implementation of DynOPS. This disadvantage has to be overcome by introducing a boolean gene for



(a) Front wheel (b) Rear wheel

Figure 3.11: Manufactured rim prototypes.

each ply that decides whether the ply is omitted or not. Even if a certain ply is removed the defining genes for the choice of material and orientation remain in the genotype and are transmitted to the next generation, although they do not influence the phenotype solution. Thus, the optimization efficiency and the solution quality might be affected adversely. The further research is therefore focused on the development of variable-length genotypes and their appropriate operators (cf. 5). Furthermore, the limited number of evaluations requires a most efficient EA to find superior solutions. A valuable method to increase the algorithm's efficiency is to investigate adaptive operator rates (cf. 4). In other words, successful operators should be applied more often than less efficient operators.

This project *"Development of High-Performance CFRP Motorcycle Rims by Using Innovative Evolutionary Optimization Strategies"* was supported by CTI (no 6469.1). Several prototype rims have been manufactured and a first race track test has been successfully conducted.

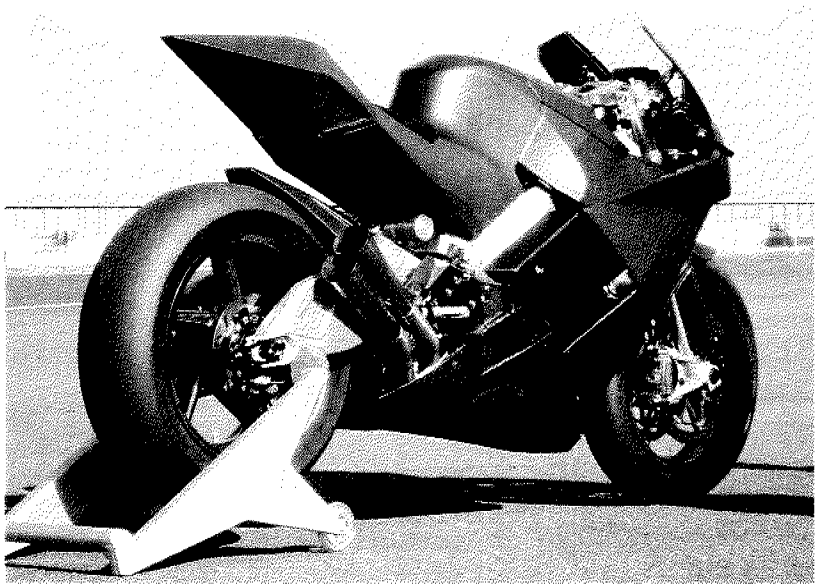


Figure 3.12: Racing motorcycle with the newly developed CFRP rims.

Chapter 4

AORCEA: Adaptive operator rate controlled Evolutionary Algorithm

This chapter is based on the paper draft submitted to Computers & Structures: *AORCEA - an Adaptive Operator Rate Controlled Evolutionary Algorithm* [100].

4.1 Introduction

During the last two decades much of research has been invested in the field of adaptive EAs. The motivation to investigate a wide variety of adaptation techniques is the appealing idea to adjust the EA to the problem while solving the problem itself.

Each EA consists of components which need to be defined. These components are variation operators such as mutation and crossover, selection mechanisms for selecting parents for reproduction and for the determination of individuals which are kept in the current population, etc. Each of these components is influenced by strategy parameters, e.g. population size, operator rates, or tournament size of selection, which have to be set. The choice of good parameter values is most often based on experience and requires therefore considerable effort.

De Jong [21] experimentally developed a heuristic for a traditional GA with one-point crossover and bit mutation which proved to be efficient for a number of numerical problems. Later studies using a meta-algorithm [23] to find suitable values or using exhaustive testing [101] led to further recommendations regarding the parameter values. Despite these recommendations there is no evidence that there exists a parameter setup which is optimal for an entire optimization run. Consequently, adapting the parameter values on the run is a promising way to improve EA performance. In Section 2.2.4 a

comprehensive classification proposed by Eiben et al. [84] and Angeline [102] considering several aspects of adaptation is presented and therefore not repeated here.

In this chapter an EA providing a robust operator setting for arbitrary optimization problems is presented. AORCEA - the Adaptive Operator-Rate Controlled Evolutionary Algorithm - can be categorized as follows: AORCEA works on the variation operator rates in an adaptive way where the changes of the strategy parameters affect the population-level and the evidence of change is based on the success of operators as well as on a relative diversity of the population with respect to the current best individual.

The next section gives a detailed description of AORCEA followed by some numerical benchmark function optimizations in Section 4.3. In Section 4.4 the performance of this novel algorithm is tested by optimizing a tubular steel trellis frame of a Ducati 998S motorcycle and Section 4.5 concludes the chapter.

4.2 AORCEA - the algorithm

The motivation for introducing an adaptive strategy arose from the recurrent problem of setting appropriate operator rates for different optimization problems.

4.2.1 The components of AORCEA

Figure 2.3 illustrates the general scheme of an adaptive EA. Theoretically, arbitrary control parameters of an EA can be adjusted in the adaptation phase, e.g., the population size, operator rates, or the fitness definitions, which directly influence the optimization performance. Within the scope of AORCEA only the operator rates of the applied variation operators are adjusted on the basis of success and relative diversity measures. The algorithm can further be characterized as follows:

Representation For structural optimization problems it is convenient to use a heterogeneous genotype (cf. Appendix A) since often different parameter types are required. A heterogeneous vector genotype of constant length is therefore employed for the numerical benchmark optimizations, where only *float-genes* are used, as well as for the structural optimization example where two different gene types, namely *float-* and *integer-genes* are assembled to the genotype. The details of the representation of the structural optimization example are discussed in Section 4.4.1.

Variation operators In general, the variation operators presented in Section 2.4, particularly suited for the constant-length and heterogeneous genotype, can be applied within the scope of AORCEA. Each of these operators

is assigned with a unique operator rate determining its probability of application during the evolutionary process. An important difference to the traditional GA [19] is that AORCEA does not sequentially apply both crossover and mutation to the selected individuals in order to produce offspring. Instead, only a single operator is chosen from the set of available operators to produce the child individual. This style of algorithm which is successfully applied in Tuson [103] and Davis [104] is used to estimate success and relative diversity measures for each operator application. Then, these measures are taken as a basis for the adaptation of the respective operator rates.

Fitness formulation For the numerical benchmark functions as well as for the optimization of the tubular steel trellis frame the fitness function formulation presented in Section 2.5 is applied. For brevity, the parameters defining the fitness functions are not stated within this thesis.

4.2.2 Adaptation strategy

Evolutionary search is always an interaction between exploration of new regions and exploitation of previously detected good regions of the search space. The adaptation strategy proposed within this chapter, schematically shown in Figure 4.1, focuses on determining whether exploitative or explorative operators should be favored in the current search state.

First, a best-fitness-frequency measure is evaluated indicating the success of the current search. Depending on this value either exploitative or explorative variation operators are forced in order to accelerate the search process. In other words, if the search process is too exploitative, i.e. the search stagnates in a local optimum, the operator rates of explorative variation operators are increased. On the other hand, if the search process is successful the rates of operators finding above-average offspring are increased in order to explore the region around the current best solution. Consequently, the next step is to determine which operators are of explorative and which are of exploitative nature. For that purpose a success and a relative diversity measure are introduced. The success measure indicates which variation operators are able to produce offspring individuals of above-average quality, whereas the relative diversity measure determines how exploitative the variation operators are. Then, the operators are ranked based on these measures and their rates are adapted, whereas the rates of operators with low rank are increased and operators having a high rank get a lower probability of application.

The subsequent sections describe the best-fitness-frequency measure, the success as well as the relative diversity measure, and the operator-rate adaptation in detail.

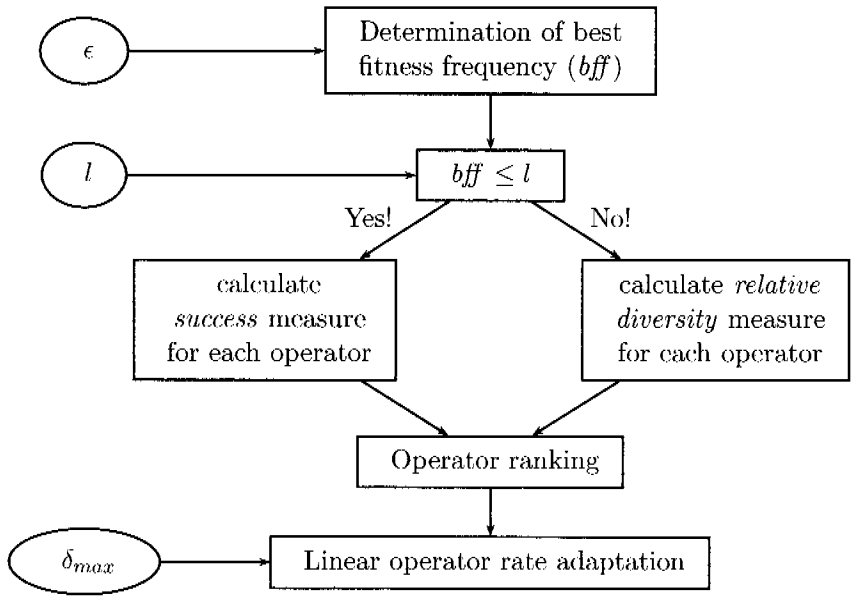


Figure 4.1: Operator rate adaptation algorithm. Input variables are shown in oval boxes.

4.2.2.1 The best-fitness-frequency measure

A popular method to investigate the search state is the analysis of the so-called fitness landscape[1] as for example shown by Puisa [105]. Although the exact shape of the fitness landscape is generally unknown for structural optimization problems, the distribution of the fitness values on the fitness landscape indicates the search state. If all the fitness values of the current population are extremely close to each other, this either indicates that the search stagnates in a at least local optimum or that the fitness landscape is rather flat. On the other hand, a population containing individuals with differing fitness values indicates a rather steep uni- or multimodal fitness landscape where the search process quite often finds improved solutions.

A simple analysis of the search state can therefore be estimated by calculating the so-called best-fitness frequency (*bff*). The *bff* value is based

[1]The fitness landscape represents a hypersurface in the (n+1)-dimensional space, where n is the number of genes in the genotype and the additional dimension stands for the fitness value. Each individual can therefore be represented as a point on this hypersurface.

on the frequency distribution of the fitness values of the current population. As shown in Figure 4.2, a histogram of the fitness values can be established. The class limits defining the class intervals are determined by f_{i^*}, i.e. the fitness value of the current best individual, and a factor ε determining the width of the class interval. The *bff* value of a single individual **i** of the current population is calculated as

$$bff_i = \begin{cases} 1, & \text{if } |f_i - f_{i^*}| \leq f_{i^*} \cdot \varepsilon \\ 0, & \text{else,} \end{cases} \tag{4.1}$$

where f_i is the fitness value of individual **i**. The total *bff* value of the current population is then calculated as

$$bff = \frac{1}{N} \sum_{i}^{N} bff_i, \tag{4.2}$$

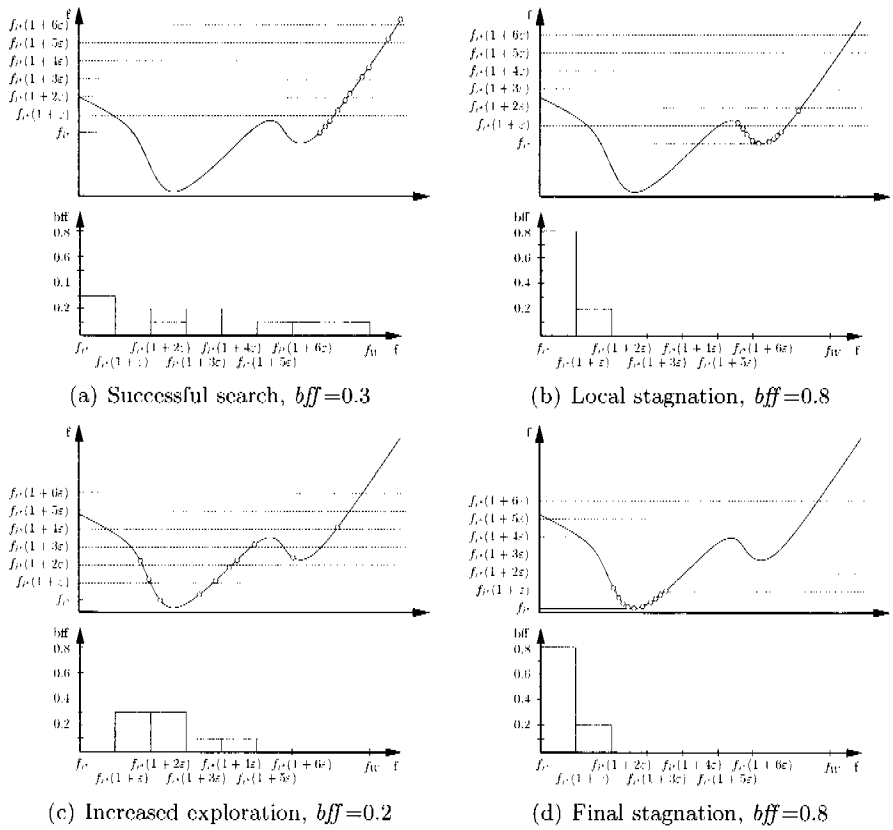
where N denotes the population size. In other words, the *bff* value corresponds to the proportion of individuals in ϵ-distance to the best individual in the population.

Figure 4.2 schematically illustrates different states of the search process with the corresponding *bff* values. A rather low *bff* value can be found in Subfigure 4.2(a) where the search is successful on a rather steep region of the fitness landscape. Only few individuals are close to the current best solution since in almost every new generation an improved individual is found. In Subfigure 4.2(b) the search reaches a local optimum and only slightly improved individuals can be found, hence, the *bff* value is increased. If the search algorithm is able to escape from this local optimum, the *bff* value is reduced again as illustrated in Subfigure 4.2(c). Finally, the search algorithm reaches another at least local optimum in Subfigure 4.2(d) and the *bff* value is increased again.

Summarizing, low *bff* values indicate a rather successful or explorative search. On the other hand, high *bff* values indicate stagnation in a at least local optimum. By introducing a threshold value $l \in [0, 1]$ defining a balance between exploitation and exploration, the adaptation strategy can be described as follows:

- $bff \leq l$: Increase the operator rates of successful operators contributing above average to the search process performance.
- $bff > l$: Increase the operator rates of explorative operators in order to escape from the local optimum by introducing more diversity into the population.

Thus, measures for success and diversity of the applied operators need to be defined.



(a) Successful search, bff=0.3

(b) Local stagnation, bff=0.8

(c) Increased exploration, bff=0.2

(d) Final stagnation, bff=0.8

Figure 4.2: Schematic illustration of different search states of an evolutionary search on the fitness landscape. Shaded areas of the histograms indicate the bff value (relative frequency of the most left class interval).

4.2.2.2 Operator success measure

In general, the evolutionary search process is assumed to be a minimization problem. Thus, the success measure for the application of an operator of type X in generation number G producing an individual **i** is defined as

$$s_{i,X}^{G} = \begin{cases} \frac{f_{pi}-f_{ci}}{f_{pi}-f_{i^*}}, & \text{if } f_{ci} \leq f_{pi} \\ 0, & \text{else} \end{cases} \tag{4.3}$$

where f_{pi} is the fitness of the best parent individual, f_{ci} is the fitness of the newly created child individual **i**, and f_{i^*} is the fitness of the overall best individual included in the current population. Thus, the maximum success

measure $s^G_{max} = 1$ occurs, if a new overall best individual ($f_{ci} = f_{i^*}$) is found. Then, for each operator of type X (e.g. one-point crossover, uniform mutation, etc.) an average success value can be calculated, i.e.

$$s^G_X = \frac{1}{N_X} \sum s^G_{i,X}, \tag{4.4}$$

where N_X is the number of operator applications of type X in the respective generation G. The operator success measure uses the overall best fitness as well as the parent fitness. Consequently, each individual needs to hold the information about its parent individual(s). This is realized by recording a MySQL database storing all the individuals on the run. For a simple numerical function optimization this may slow down the optimization process, but for a structural optimization problem, where one evaluation often takes seconds to minutes, the additional "bookkeeping" does not influence the performance significantly. In contrast, it is an advantage to have access to all the evaluated individuals.

4.2.2.3 Relative operator diversity measure

The relative operator diversity measure analyzes the distribution of the individuals in the search space with respect to the current best individual. The Euclidean distance of an individual **i** originating from the application of an operator of type X in generation G to the current best individual **i*** is calculated

$$d^G_{i,X} = \sqrt{\sum_{j=1}^{n} (i_j - i^*_j)^2}, \tag{4.5}$$

where n is the number of genes of the genotype, and i_j and i^*_j are the components of the respective genotypes. A weighted Euclidean distance could alternatively be used of the form

$$dw^G_{i,X} = \sqrt{\sum_{j=1}^{n} w_j (i_j - i^*_j)^2}, \tag{4.6}$$

where w_j is a weighting factor for each component of the genotype. This weighting factor could serve the purpose of mapping the current gene values into intervals [0,1], but then the lower as well as the upper limit of the respective gene need to be known. Since the heterogeneous genotype concept does not generally require lower or upper limits for each gene, the gene values can theoretically grow to infinity. Thus, it is not always possible to define lower and upper limits for a gene and therefore the weighting factor could not be determined and a workaround would have to be found. As long

as the genotype consists of identical genes, e.g. for typical numerical benchmark functions, such weighting factors have no influence on the optimization performance.

However, within this work no weighting factors are used and the relative diversity measure therefore analyzes at which absolute distance to the current best individual new individuals are created. The relative diversity measure for the application of an operator of type X in generation number G producing an individual i is defined as

$$t_{i,X}^{G} = \begin{cases} 0, & \text{if } f \leq f_{i^*}(1+\varepsilon) \\ \frac{1}{1+\frac{d_{i,X}^{G}}{d_{max}^{G}} \cdot a}, & \text{if } f > f_{i^*}(1+\varepsilon), \end{cases} \tag{4.7}$$

where d_{max}^{G} is the maximum Euclidean distance between the current best individual and all individuals of the current population. The factor a is introduced for scaling purposes only and is set to $a = 9$. Operators producing individuals with fitness values close to the current best individual are rated with a relative diversity measure of 0, since these individuals do not essentially help to escape from this region. It might happen that individuals with similar fitness values are located in a different region of the search space, but for typical structural optimization problems this is unlikely, i.e., this situation occurs only rarely and therefore has a minor influence on the optimization performance. All other individuals are assigned with relative diversity measure values decreasing slightly with increasing Euclidean distance to the current best individual. This way, operators exploring the region around the occupied region can be identified. Again, for each operator type X an average relative diversity value can be calculated for each generation G, i.e.

$$t_{X}^{G} = \frac{1}{N_X} \sum t_{i,X}^{G}, \tag{4.8}$$

where N_X is the number of operator applications of type X.

4.2.2.4 Operator rate adaptation

The aim of the operator rate adaptation is to provide operator rates guaranteeing a balance between exploration and exploitation. Depending on the current *bff* value either the success or the relative diversity measure is applied. If the current *bff* value is below the target *bff* value l, the success ranking is employed in order to force the successful operators, see Figure 4.3. On the other hand, the relative diversity measure is applied if the search stagnates and finds a lot of similar individuals close to the current best solution.

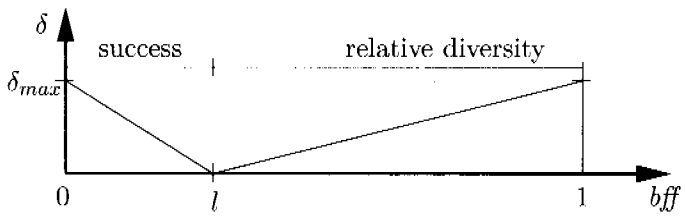


Figure 4.3: Adaptation rate based on *bff* measure.

An adaptation rate δ is introduced depending on the difference between the current *bff* value and the threshold value l.

$$\delta = \begin{cases} -\frac{\delta_{max}}{l}(bff - l), & \text{if } bff \leq l \\ \frac{\delta_{max}}{1-l}(bff - l), & \text{if } bff > l, \end{cases} \tag{4.9}$$

By setting the target value l and the maximum adaptation value δ_{max} the proportion between exploration and exploitation as well as the magnitude of adaptation can be controlled.

Figure 4.4 illustrates the linear adaptation of the operator rates which always sum up to a total of 100%. On the left, the current rates of six operators are shown. Then, the operators are ranked according to the previously determined success or relative diversity measure. The operator fulfilling the respective measure the best is assigned with rank 1, the worst operator is assigned with rank 6. Subsequently, the operator rates are linearly in- or decreased for the next generation according to

$$p_X^{G+1} = p_X^G + \left[\frac{N_O + 1 - 2R_X}{N_O - 1}\right]\delta, \tag{4.10}$$

where G is the current generation, N_O is the total number of applied operators, and R_X is the rank of the operator of type X. Furthermore, a minimum operator rate p_{min} is guaranteed for each operator in order to prevent that single operators are completely excluded from the optimization process.

Practice has shown that not only the measures of the current generation should be considered to rank the operators but also some previous generations should be taken into account. If a lot of previous generations are considered, this may not reflect the operator performances of the current search state. On the other hand, only considering the operator performances of the current iteration may be statistically poorly founded. Therefore, the last five generations are averaged to success and relative diversity measures for all operators and the adaptation of the operator rates is performed each fifth generation.

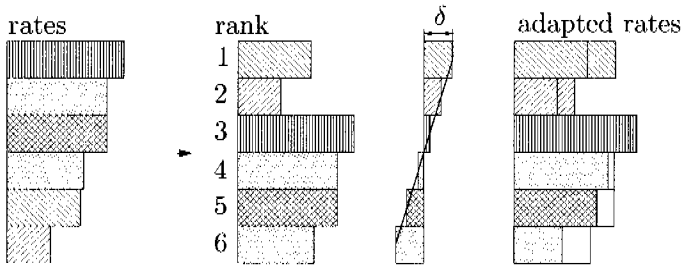


Figure 4.4: Linear operator rate adaptation.

4.3 Numerical examples

A series of benchmark functions is investigated to compare the performance of AORCEA with a non-adaptive EA. The general setup of the non-adaptive EA is exactly the same as it is for AORCEA in terms of population size, number of generations, selection, etc., with the only exception of constant operator rates. For the investigations at hand four different variation operators are chosen, namely one-point crossover (0.35), uniform crossover (0.35), uniform mutation (0.15), and Gaussian mutation (0.15). The initial operator rates are given in parentheses which remain constant for the non-adaptive optimization.

For each optimization problem the population size is $P = 2n$, where n is the problem dimensionality, and the maximum number of generations is set to $G_{max} = 20n$ for unconstrained and $G_{max} = 40n$ for constrained benchmark functions. The ε-value from Equation 4.1 determining the *bff* value is set to 0.01 and the minimum operator rate p_{min} is set to 10% for each operator.

The sensitivity of AORCEA with respect to the parameter values of the maximum adaptation δ_{max} and the target *bff* value l from Equation 4.9 is investigated by using different combinations of these adaptation parameter values listed in Table 4.1. The setups S_1 to S_3 rather slowly adapt to the search state since the magnitude of adaptation δ_{max} is relatively small. The setups S_4 to S_7 are characterized by increased δ_{max} values and therefore adapt faster. Finally, the different l values should investigate which proportion of exploration and exploitation performs better.

A total of seven well-known unconstrained and four constrained functions listed in Tables 4.2 and 4.3, respectively are examined. The test functions F1 - F3 are from De Jong's [21] test function suite. The further unconstrained functions are Ackley's function [106] (F4), a version of Griewank's function [107] (F5), the Michalewicz function [46] (F6), and the Rastrigin function from [108] (F7). The constrained functions C1 and C2 are taken from Floudas and Pardalos [109], C3 and C4 are from Hock and

Setup	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7
δ_{max}	0.0	0.05	0.05	0.05	0.1	0.1	0.2	0.2
l	-	0.1	0.25	0.5	0.1	0.25	0.1	0.25
ε	-	0.01	0.01	0.01	0.01	0.01	0.01	0.01

Table 4.1: Test function setups defined by adaptation parameters. Setup S_0 is the reference optimization without adaptation.

F	n	function	range of x_i
F1	10	$f(n,\mathbf{x}) = \sum_{i=1}^{n} x_i^2$	$x_i \in [-10.0, 10.0]$
F2	20	$f(n,\mathbf{x}) = \sum_{i=1}^{n-1} 100(x_i - x_{i-1}^2)^2 + (1 - x_{i-1})^2$	$x_i \in [-10.0, 10.0]$
F2a	50	$f(n,\mathbf{x}) = \sum_{i=1}^{n-1} 100(x_i - x_{i-1}^2)^2 + (1 - x_{i-1})^2$	$x_i \in [-10.0, 10.0]$
F3	5	$f(n,\mathbf{x}) = 30 + \sum_{i=1}^{n} \lfloor x_i \rfloor$	$x_i \in [-5.12, 5.12]$
F4	10	$f(n,\mathbf{x}) = -20\,e^{-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n} cos(2\pi x_i)} + 20 + e$	$x_i \in [-32.768, 32.768]$
F5	10	$f(n,\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{n}(x_i - 100)^2 - \prod_{i=1}^{n} cos\left(\frac{x_i - 100}{\sqrt{i+1}}\right) + 1$	$x_i \in [-600.0, 600.0]$
F6	10	$f(n,\mathbf{x}) = -\sum_{i=1}^{n}\left(sin(x_i)\, sin^{20}\left(\frac{ix_i^2}{\pi}\right)\right)$	$x_i \in [0, \pi]$
F7	20	$f(n,\mathbf{x}) = 10n + \sum_{i=1}^{n}\left[x_i^2 - 10\,cos(2\pi x_i)\right]$	$x_i \in [-5.12, 5.12]$
F7a	50	$f(n,\mathbf{x}) = 10n + \sum_{i=1}^{n}\left[x_i^2 - 10\,cos(2\pi x_i)\right]$	$x_i \in [-5.12, 5.12]$

Table 4.2: Unconstrained benchmark functions.

Schittkowski [110].

4.3.1 Results

For each test function and each setup a total number of 20 independent runs were performed to get statistically well-founded data. The numerous optimization runs produced a huge amount of data, too much to discuss them all in detail. Nevertheless, in order to compare the optimization performances of the different setups, Wilcoxon rank-sum tests [111] are used to determine whether the adaptive setups are significantly superior to the non-adaptive setup.

The Wilcoxon rank-sum test is a statistical significance test for assessing whether the difference in medians between two samples of observations is statistically significant, whereas the null hypothesis is that the two samples have a common median. Here, the first sample consists of the best fitness values of the 20 runs after the maximum number of generations of the non-

C	n	function / constraints
C1	13	$f(\mathbf{x},\mathbf{y}) = 5x_1 + 5x_2 + 5x_3 + 5x_4 - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=1}^{9} y_i$
		$2x_1 + 2x_2 + y_6 + y_7 \leq 10$ $\quad 2x_1 + 2x_3 + y_6 + y_8 \leq 10$
		$2x_2 + 2x_3 + y_7 + y_8 \leq 10$ $\quad -8x_1 + y_6 \leq 0$
		$-8x_2 + y_7 \leq 0$ $\quad -8x_3 + y_8 \leq 0$
		$-2x_4 - y_1 + y_6 \leq 0$ $\quad -2y_2 - y_3 + y_7 \leq 0$
		$-2y_4 - y_5 + y_8 \leq 0$ $\quad 0 \leq x_i \leq 1, i = 1,2,3,4$
		$0 \leq y_i \leq 1, i = 1,2,3,4,5,9$ $\quad 0 \leq y_i, i = 6,7,8$
C2	6	$f(\mathbf{x},y) = -10.5x_1 - 7.5x_2 - 3.5x_3 - 2.5x_4 - 1.5x_5 - 10y - 0.5\sum_{i=1}^{5} x_i^2$
		$6x_1 + 3x_2 + 3x_3 + 2x_4 + x_5 \leq 6.5$ $\quad 10x_1 + 10x_3 + y \leq 20$
		$0 \leq x_i \leq 1$ $\quad 0 \leq y$
C3	7	$f(\mathbf{x}) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7$
		$127 - 2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 \geq 0$ $\quad 282 - 7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 \geq 0$
		$196 - 23x_1 - x_2^2 - 6x_6^2 + 8x_7 \geq 0$ $\quad -4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7 \geq 0$
		$-10.0 \leq x_i \leq 10.0, i = 1,\ldots,7$
C4	10	$f(\mathbf{x}) = x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2 + 2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_10 - 7)^2 + 45$
		$105 - 4x_1 - 5x_2 + 3x_7 - 9x_8 \geq 0$ $\quad -3(x_1 - 2)^2 - 4(x_2 - 3)^2 - 2x_3^2 + 7x_4 + 120 \geq 0$
		$-10x_1 + 8x_2 + 17x_7 - 2x_8 \geq 0$ $\quad -x_1^2 - 2(x_2 - 2)^2 + 2x_1x_2 - 14x_5 + 6x_6 \geq 0$
		$8x_1 - 2x_2 - 5x_9 + 2x_10 + 12 \geq 0$ $\quad -5x_1^2 - 8x_2 - (x_3 - 6)^2 + 2x_4 + 40 \geq 0$
		$3x_1 - 6x_2 - 12(x_9 - 8)^2 + 7x_10 \geq 0$ $\quad -0.5(x_1 - 8)^2 - 2(x_2 - 4)^2 - 3x_5^2 + x_6 + 30 \geq 0$
		$-10.0 \geq x_i \geq 10.0, i = 1,\ldots,10$

Table 4.3: Constrained benchmark functions.

adaptive setup. The second sample contains the best fitness values of an adaptive setup, hence, each adaptive setup is compared to the non-adaptive setup by means of a rank-sum test. The p-value returns the significance for testing the null hypothesis. The alternative is that the median of one sample is shifted from the median of the other sample by a non-zero amount. Thus, for p-values below 5% a significant difference of the median values is much likely.

4.3.1.1 Unconstrained benchmark functions

The results presented in tables 4.4 and 4.5 show the best and the median objective values after $20n$ generations as well as the Wilcoxon rank-sum test results for the optimization setups given in Table 4.1. Obviously, not all of the optimizations are fully converged to their global optima, but nevertheless an interpretation of the optimization results is possible. The test function results are briefly discussed:

- F1: The median of the non-adaptive configuration (S_0) is clearly greater than the median values of all adaptive setups and only one

best value of the adaptive setups (S_1) is slightly worse. Additionally, the p-value of the Wilcoxon rank-sum test is clearly below 5% for all adaptive setups indicating that they perform significantly better than the non-adaptive setup.

- F2: For both, the test functions F2 and F2a, the adaptive setups are again superior to the non-adaptive setup because the median as well as the best values are below the reference values of the setup S_0. However, for function F2 the setup $S3$ cannot be considered as significantly better and for function F2a the setups S_2, S_3, S_5, and S_7 miss the 5% criterion, but they are never significantly worse than the reference setup S_0.

- F3: The step function is difficult to be interpreted. Each setup finds the global optimum at least once, whereas the fast adapting setups (S_1 to S_7) more often solve the problem since even their median values coincide with the global optimum.

- F4: Similar to F1, the adapting setups show the better performance, i.e., their median values are clearly below the reference of setup S_0. Furthermore, the p-values of the rank-sum test confirm this observation, they are all clearly below the 5% significance level. The convergence behavior of the different strategies are shown in Figure 4.5(a).

- F5: The median values as well as the p-values reveal that fast adaptation leads to a better convergence behavior for this benchmark function. Although all adaptive strategies show a lower median value than the non-adaptive one, the setups S_1 and S_3 slightly miss the 5% significance level. In Figure 4.5(b) the convergence behavior of the different strategies are shown.

- F6: Again, all adaptive setups perform better than the reference setup S_0 in terms of the median values. However, only the setups S_4 and S_6 reach the 5% significance level.

- F7: Although the median values of the adaptive setups are generally superior to the non-adaptive setup, the significance level is only rarely fulfilled. In particular the 50-dimensional benchmark function shows no significant differences in terms of the optimization performance, but the non-adaptive setup is never significantly better than the adaptive setups.

4.3.1.2 Constrained benchmark functions

The results presented in tables 4.6 and 4.7 analogously show the best and the median objective values after $40n$ generations for the constrained test

	S_0		S_1		S_2		S_3	
F	median	best	median	best	median	best	median	best
F1	0.14702	0.02521	0.065138	0.026211	0.086848	0.022536	0.058401	0.019852
F2	487.11	201.45	261.4	155.62	289.46	130.77	294.35	96.888
F2a	199.1	109.66	138.89	56.756	160.18	72.302	177.44	95.893
F3	1	0	1	0	1	0	1	0
F4	4.1241	1.7282	3.4377	1.6677	3.3178	2.7353	3.4691	2.4690
F5	0.23172	0.082854	0.15872	0.047653	0.15949	0.094708	0.17759	0.080719
F6	-7.7739	-8.8138	-8.1687	-9.1023	-8.2884	-9.1929	-8.2319	-9.4415
F7	78.192	54.029	67.613	45.946	70.062	42.211	75.997	35.735
F7a	287.13	212.66	282.96	244.39	276.18	202.39	291.65	206.17
	S_4		S_5		S_6		S_7	
F1	0.04524	0.014096	0.042572	0.005285	0.042878	0.016557	0.046939	0.016557
F2	305.29	122.83	254.15	104.97	261.1	156.78	221.04	105.07
F2a	125.21	87.379	152.44	51.56	146.52	56.837	151.28	45.583
F3	0	0	0	0	0	0	0	0
F4	3.1302	2.1135	3.0833	1.6939	3.1606	1.6684	2.7102	2.2463
F5	0.15339	0.057408	0.16099	0.058149	0.12516	0.042378	0.12604	0.050569
F6	-8.4073	-9.044	-8.2368	-8.7635	-8.3065	-8.7469	-7.9143	-9.1131
F7	64.886	50.821	69.566	54.50	75.37	43.425	71.325	50.322
F7a	279.63	215.26	281.12	239.38	283.68	223.46	290.39	253.01

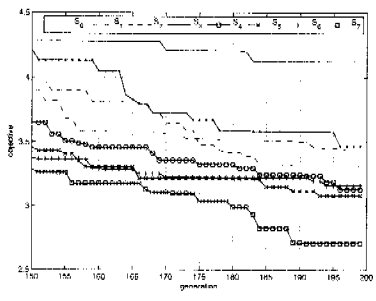
Table 4.4: Median and best values of the objective of the unconstrained benchmark functions after $20n$ generations. The global optimum $F(\mathbf{x}^*)$ of all functions is 0.0 except for $F6(\mathbf{x}^*) = -9.66$.

F	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7
F1	1.0	$4.493e^{-4}$	$4.549e^{-3}$	$4.045e^{-3}$	$1.890e^{-4}$	$1.890e^{-4}$	$1.401e^{-4}$	$2.931e^{-4}$
F2	1.0	$3.656e^{-2}$	$4.004e^{-2}$	$1.004e^{-1}$	$4.559e^{-3}$	$2.509e^{-2}$	$8.034e^{-3}$	$4.045e^{-3}$
F2a	1.0	$1.688e^{-2}$	$5.222e^{-2}$	$1.913e^{-1}$	$1.524e^{-2}$	$3.317e^{-1}$	$4.549e^{-3}$	$1.353e^{-1}$
F3	1.0	$4.671e^{-1}$	$5.491e^{-1}$	$9.340e^{-1}$	$3.398e^{-1}$	$3.258e^{-1}$	$4.248e^{-2}$	$8.300e^{-2}$
F4	1.0	$4.493e^{-4}$	$1.713e^{-3}$	$4.044e^{-2}$	$1.890e^{-4}$	$5.934e^{-4}$	$1.204e^{-4}$	$1.401e^{-4}$
F5	1.0	$1.084e^{-1}$	$2.509e^{-2}$	$1.353e^{-1}$	$1.688e^{-2}$	$1.524e^{-2}$	$3.185e^{-3}$	$1.941e^{-3}$
F6	1.0	$1.560e^{-1}$	$6.735e^{-2}$	$8.592e^{-2}$	$2.762e^{-2}$	$7.313e^{-2}$	$2.509e^{-2}$	$3.134e^{-1}$
F7	1.0	$3.036e^{-2}$	$1.084e^{-1}$	$9.296e^{-2}$	$2.063e^{-2}$	$1.237e^{-2}$	$1.084e^{-1}$	$5.222e^{-2}$
F7a	1.0	$9.702e^{-1}$	$5.691e^{-2}$	$2.471e^{-1}$	$2.179e^{-1}$	$9.108e^{-1}$	$8.227e^{-1}$	$9.404e^{-1}$

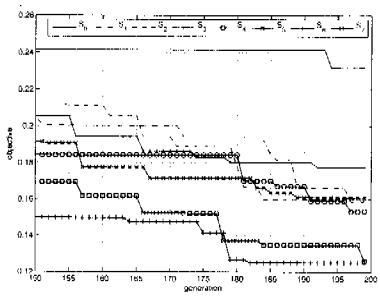
Table 4.5: The p-values of the Wilcoxon rank-sum test results for the unconstrained benchmark functions. Values above the 5% significance level are printed in italic.

functions. Due to the fact that the fitness is a weighted sum of the objective and the numerous constraints it is necessary to define how the best individual is determined. In this case, it is not the individual with the lowest objective value, but it is the individual with the lowest overall fitness value.

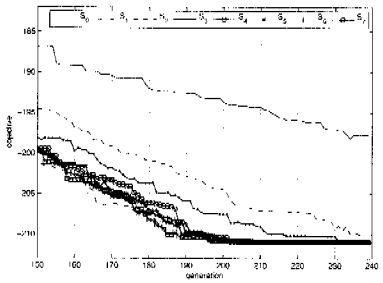
- C1: After 520 generations the search is converged for all strategies, hence, the median as well as the best values seem to be close to each other. However, the p-value of the rank-sum test clearly reveals that the adaptive strategies perform significantly superior compared to the



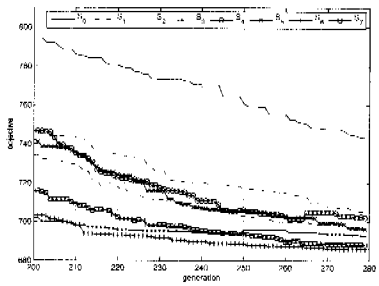
(a) F4: Ackley's function



(b) F5: Griewank's function



(c) C2: Floudas' function



(d) C3: Hock's function

Figure 4.5: Median values for different adaptation strategies compared to the non-adaptive strategy.

non-adaptive setup.

- C2: For this constrained test function the adaptive strategies work extremely efficient, since they all converge to a solution close to the global optimum. The best solutions are all below -212.16 and the median values are smaller than -209.96, see Figure 4.5(c). At the same point of the search process the non-adaptive strategy finds a best individual with an objective value of -210.93 and the median objective value is only -197.69. Obviously, also the p-values confirm the superiority of the adaptive strategies.

- C3: The convergence behavior depicted in Figure 4.5(d) clearly shows that the adaptive strategies again outperform the non-adaptive strategy. The adaptation of the operator rates is obviously able to cope with optimization tasks searching along constraints.

- C4: Finally, the adaptive strategies are again better than the non-adaptive setup. The best solutions and the median objective value of

each adaptive setup are below the reference values of S_0 and only the setup S_3 does not reach the 5% significance level.

	S_0		S_1		S_2		S_3	
F	median	best	median	best	median	best	median	best
C1	-13.922	-13.958	-13.944	-13.943	-13.95	-13.948	-13.939	-13.955
C2	-197.69	-210.93	-210.95	-212.69	-209.96	-212.16	-210.82	-212.74
C3	743.32	691.90	705.52	682.95	696.20	683.05	692.91	683.81
C4	39.939	32.803	33.406	28.638	34.449	27.552	35.266	29.350
	S_4		S_5		S_6		S_7	
C1	-13.955	-13.954	-13.952	-13.954	-13.942	-13.945	-13.948	-13.949
C2	-210.94	-212.20	-210.94	-212.92	-210.94	-212.21	-210.94	-212.25
C3	702.22	683.17	696.35	683.00	685.99	683.27	688.26	683.05
C4	33.956	27.319	32.515	27.654	32.431	27.322	34.743	29.307

Table 4.6: Median and best values of the objective of the constrained benchmark functions after $40n$ generations. The global optima are: $C1(\mathbf{x}^*) = -15.0$, $C2(\mathbf{x}^*) = -213.0$, $C3(\mathbf{x}^*) = 680.6300573$, and $C4(\mathbf{x}^*) = 24.3062091$.

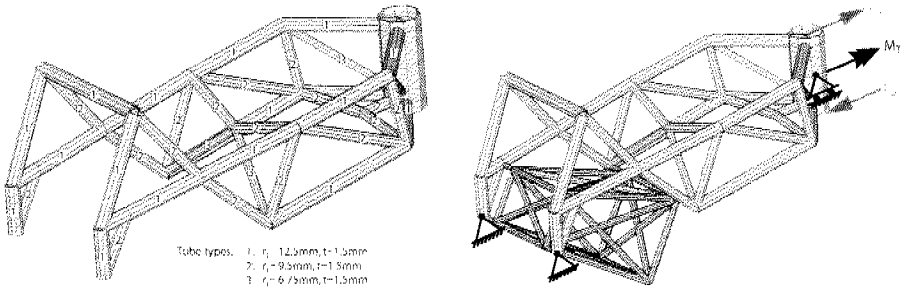
F	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7
C1	1.0	$8.844e^{-5}$	$1.033e^{-4}$	$1.204e^{-4}$	$1.033e^{-4}$	$1.204e^{-4}$	$1.399e^{-4}$	$8.857e^{-5}$
C2	1.0	$1.964e^{-4}$	$4.045e^{-3}$	$1.237e^{-2}$	$4.045e^{-3}$	$3.981e^{-4}$	$1.823e^{-3}$	$1.087e^{-3}$
C3	1.0	$5.111e^{-3}$	$1.688e^{-2}$	$4.785e^{-2}$	$5.733e^{-3}$	$8.034e^{-3}$	$1.324e^{-3}$	$1.033e^{-4}$
C4	1.0	$1.204e^{-4}$	$6.424e^{-3}$	*$1.560e^{-1}$*	$5.111e^{-3}$	$3.902e^{-4}$	$6.424e^{-3}$	$1.524e^{-2}$

Table 4.7: The p-values of the Wilcoxon rank-sum test results for the constrained benchmark functions. Values above the 5% significance level are printed in italic type.

Summarizing, for each constrained and unconstrained test function the adaptive strategies show superior median values. The only exception is setup S_7 of function F7a, where the median value of the non-adaptive setup is better. Furthermore, more than 50% of the adaptive setups of the unconstrained functions and almost 100% of the adaptive setups of the constrained functions perform significantly superior to the non-adaptive indicated by the Wilcoxon rank-sum test. Although there are some setups which are not significantly better, they still perform on at least the same level as the traditional EA. Unfortunately, it is impossible to recommend a parameter setup for δ_{max} and l since none of the investigated setups clearly outperforms the other setups.

4.4 Tubular steel trellis frame

AORCEA is particularly developed for structural optimization tasks based on the heterogeneous genotype. Thus, an example for such a structural optimization is presented here by means of the tubular steel trellis frame of a Ducati 998S motorcycle.



(a) Original tube dimensions (b) Load case with simplified engine block

Figure 4.6: FE-model of the tubular steel trellis frame of a Ducati 998S motorcycle.

4.4.1 Parameterization

The symmetric FE-model depicted in Figure 4.6(a) shows the original tube dimensions. A total of 15 tube dimensions defined by inner radius and thickness are subject to optimization, whereas identical tubes on each side of the frame are defined by the same parameter pair. The steering shaft tube dimensions and the positions of all tubes remain unchanged, hence, no functional or manufacturing constraints are violated. Two different parameterizations are examined:

- *free tubes:* This parameterization allows individual inner radii and thicknesses for all 15 different tubes leading to a genotype consisting of 30 float-genes.
- *custom tubes:* Only three custom tube types are permitted whose inner radii and thicknesses are defined by six float-genes. One of these three custom tubes is to be chosen for each of the tubes what is realized with 15 integer-genes assigning the first, second, or third custom tube.

4.4.2 Optimization formulation

After mapping the genotype to the FE-model depicted in Figure 4.6(b) two load cases are simulated in order to estimate the quality of a given solution.

The FE-model is composed of standard 2-node beam elements and incorporates the engine block which obviously influences the stiffness behavior of the entire system. The frame is clamped at the connection points to the swing arm and at an additional support at the headset, see Figure 4.6(b). First, the torsional stiffness is evaluated by applying a torsional moment M_T at the headset and solving a linear static analysis. A stiffness measure S is defined with the applied torsional moment and the resulting rotation λ_T between the swing arm axis and the headset axis:

$$S = \frac{M_T}{\lambda_T} \quad \left[\frac{\text{Nm}}{^\circ}\right]. \tag{4.11}$$

The second load case simulates braking by applying a pair of forces F_B at the headset. After a second linear static analysis the maximum Von Mises stress σ_{max} is evaluated based on the nodal stress values.

The objective of this optimization is to reduce the total mass of the frame in compliance with the torsional stiffness and mechanical stress constraints.

$$\begin{aligned} \text{Minimize} \quad & m = \sum_{i=1}^{E} m_i \\ \text{subject to} \quad & S = 1330 \pm 5 \left[\frac{\text{Nm}}{^\circ}\right] \\ & \sigma_{max} \leq 470.0 \ [\text{MPa}], \end{aligned} \tag{4.12}$$

where E is the number of finite elements and m_i the mass of element number i. The torsional stiffness is formulated as a target constraint which should be exactly met and the mechanical stresses must remain below the critical limit. The fitness function implementation is based on a weighted sum of ratings of the objective and both constraints (cf. Section 2.5).

The same seven adaptive setups are compared to the non-adaptive strategy defined in Table 4.1 by performing 20 runs for each setup. Two different operator configurations are investigated.
This configuration is denoted as free$^\alpha$ and custom$^\alpha$, respectively, and the minimum operator rate p_{min} is analogously set to 10%. The second operator configuration (free$^\beta$ and custom$^\beta$) includes some more variation operators, i.e. one-point, two-point, uniform, intermediate, segment, and hypercube crossover (0.1167), and uniform, deterministic uniform, Gaussian, and deterministic Gaussian mutation (0.075), whereas the minimum operator rate p_{min} is reduced to 5%. A brief definition of these genetic operator types is given in Section 2.4.

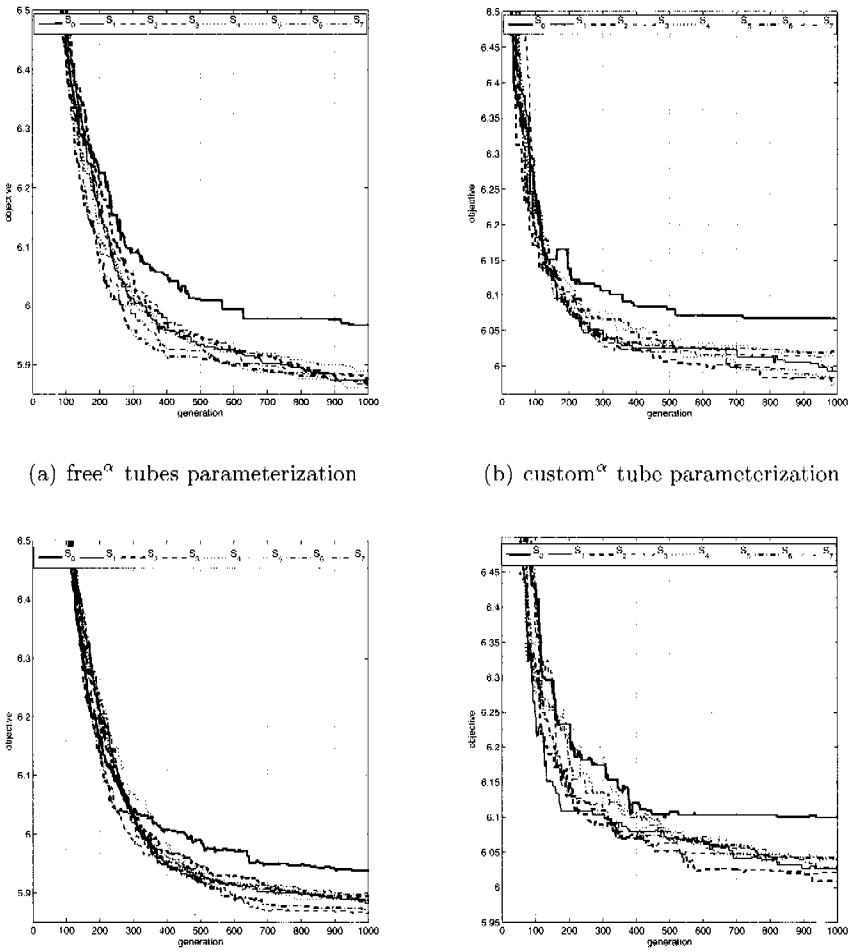
Furthermore, the population size for both, the free and the custom tube parameterization, is chosen to be 60 and the maximum number of generations is limited to 1000. In tables 4.8 and 4.9 the results of both parameterizations are presented for all setups and the convergence plots are depicted in Figure 4.7.

4.4.3 Results

For the operator configurations free$^{\alpha}$ and custom$^{\alpha}$ no significant differences between the convergence behavior of the adaptive and the non-adaptive setups can be seen during the first 100 to 200 generations. Presumably this behavior originates from the fact that the adaptive algorithm needs some generations in the beginning to identify efficient operators. Once these operators are detected their rates are increased and contribute to a superior convergence behavior. AORCEA seems to be quite insensitive to the adaptation parameters since all adaptive setups show approximately the same performances which are clearly superior to the non-adaptive setup. This observation is also confirmed by the p-values of the Wilcoxon rank-sum test listed in Table 4.9. All adaptive setups produce significantly lower median values indicated by p-values below the significance level of 5%. For the custom tubes parameterization the non-adaptive setup cannot find a best solution below 6.022 kg, whereas all adaptive strategies locate solutions below 5.952 kg without violating the torsional stiffness or the mechanical stress constraint.
For the operator configurations free$^{\beta}$ and custom$^{\beta}$ the effect of adaptation occurs after approximately generation 400. Since there are much more operators involved and the minimum operator rates are relatively high, the adaptation cannot exert that much influence on the optimization performance. Nevertheless, all adaptive strategies are still superior compared to the non-adaptive strategy in terms of the best and the median fitness values. Additionally, all but one setup (S_5) are significantly better for this parameter configuration.

The comparison between the configurations free$^{\alpha}$ and free$^{\beta}$ reveals that the application of more operators helps the non-adaptive strategy to perform better. The converse behavior can be observed by comparison of the configurations custom$^{\alpha}$ and custom$^{\beta}$ where the application of more variation operators leads to generally worse optimization performances in terms of the median as well as the best solutions. All these results confirm the observations from the numerical examples in Section 4.3 where the adaptive strategies always perform superior or at least equal to the traditional algorithm.

Surprisingly, the adaptation parameters seem to have a minor influence on the optimization performance. All adaptive optimization setups show a similar convergence behavior although their adaptation parameters are different. The convergence plots in Figure 4.7 as well as the optimization performance measures in Table 4.9 confirm this observation. However, further investigations have shown that these adaptation parameters cannot be chosen arbitrarily. Other optimization runs with higher values for ε, δ_{max}, and l provide convergence behavior and best solutions which are extremely



(a) free$^{\alpha}$ tubes parameterization

(b) custom$^{\alpha}$ tube parameterization

(c) free$^{\beta}$ tubes parameterization

(d) custom$^{\beta}$ tube parameterization

Figure 4.7: Convergence behavior of different setups for the tubular steel trellis frame optimization. The plot shows the median objective value of each setup.

worse compared to the non-adaptive algorithm. Thus, a reasonable choice of the adaptation parameters in the range of the presented seven setups is definitely necessary.

	S_0		S_1		S_2		S_3	
tubes	median	best	median	best	median	best	median	best
free$^{\alpha}$	5.9668	5.8095	5.8738	5.7921	5.8771	5.8086	5.8658	5.7836
custom$^{\alpha}$	6.0666	6.0222	5.9962	5.9315	5.9811	5.9518	5.9845	5.9373
free$^{\beta}$	5.9390	5.8303	5.8835	5.7993	5.8953	5.8126	5.8657	5.8028
custom$^{\beta}$	6.1001	6.0353	6.0274	5.9527	6.0095	5.9402	6.0214	5.9498
tubes	S_4		S_5		S_6		S_7	
free$^{\alpha}$	5.8898	5.8031	5.8597	5.7895	5.8644	5.8084	5.8690	5.8052
custom$^{\alpha}$	5.9746	5.9332	6.0152	5.9470	6.0222	5.9465	5.9922	5.9542
free$^{\beta}$	5.8883	5.7991	5.8893	5.7919	5.8716	5.7836	5.8971	5.7918
custom$^{\beta}$	6.0407	5.9497	6.0386	5.9583	6.0422	5.9493	6.0256	5.9389

Table 4.8: Masses in kilograms of the best and median solutions of the tubular steel trellis frame optimization.

tubes	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7
free$^{\alpha}$	1.0	$1.507e^{-3}$	$2.535e^{-4}$	$8.918e^{-4}$	$1.018e^{-3}$	$8.918e^{-4}$	$7.795e^{-4}$	$5.934e^{-4}$
custom$^{\alpha}$	1.0	$1.628e^{-4}$	$8.857e^{-5}$	$1.401e^{-4}$	$5.167e^{-4}$	$1.204e^{-4}$	$3.384e^{-4}$	$5.934e^{-4}$
free$^{\beta}$	1.0	$3.036e^{-2}$	$7.189e^{-3}$	$5.167e^{-4}$	$2.820e^{-3}$	*$6.735e^{-2}$*	$2.820e^{-3}$	$2.203e^{-3}$
custom$^{\beta}$	1.0	$5.934e^{-4}$	$2.190e^{-4}$	$1.890e^{-4}$	$6.806e^{-4}$	$1.112e^{-2}$	$7.189e^{-3}$	$4.493e^{-4}$

Table 4.9: The p-values of the Wilcoxon rank-sum test results for the tubular steel trellis frame. Values above the 5% significance level are printed in italic type.

4.5 Conclusion

AORCEA is tested on eleven well-known unconstrained and constrained numerical functions and compared to the performance of the non-adaptive strategy. For each of the test functions AORCEA performs at least slightly but most often significantly improved compared to the traditional algorithm. Additionally, AORCEA is applied to a structural optimization problem which reduces the mass of a steel trellis frame of a Ducati motorcycle while complying with the given stiffness and stress constraints. The adaptive strategy clearly outperforms the non-adaptive strategy in terms of convergence behavior and solution quality. Unfortunately, the results cannot identify an adaptation parameter setup which is clearly superior, but at least one can conclude that the performance of AORCEA is not extremely dependent on these parameters.

It is in the nature of EA development that there are always a lot of parameters decisively determining the optimization performance. Furthermore, these algorithms can only be tested on a limited set of test functions or concrete problems from engineering practice. It is therefore necessary to develop robust algorithms which are general in their nature to make them

usable for a wide variety of optimization tasks. In the future, AORCEA will be used for structural optimization problems based on the heterogeneous genotype used in this work. Further steps in the development could be the incorporation of self-adaptive strategies for some particular variation operators. As an example, the standard deviation determining the behavior of the Gaussian mutation operator seems to have a crucial influence on the optimization performance in particular when it comes to the fine tuning towards the end of the optimization run. Some self-adaptive strategies are known but mostly they are developed for algorithms using the bit representation. It is a promising approach to combine these self-adaptive approaches with AORCEA in order to increase the algorithms efficiency again.

AORCEA has not been compared to other optimization algorithms yet, but further research should address this issue in order to assess the quality of the newly developed algorithm.

Chapter 5

Variable-length representation for composite laminate optimization

5.1 Introduction

5.1.1 Research in the field of variable-length representations

On the first sight the generalization from a constant-length to a variable-length representation allowing for an almost arbitrary number of design entities seems to be a small step to a generalized representation concept. However, the additional freedom gained by this generalization is paid with an important restriction. The variation operators, in particular the crossover operators, need to be modified in order to exclusively produce feasible solutions. As an example, for the heterogeneous genotype concept presented in Appendix A only genes of the same type encoding similar phenotype entities are allowed to be mated, otherwise genotypes would be produced which could not be mapped to reasonable phenotypes.

Numerous researchers have already investigated several variable-length representation concepts. An early study of variable-length representation is the messy GA [112, 113] which is based on a binary genotype where the gene is an ordered pair *(locus, value)*. Although the number of bits required to generate a solution is constant, the number and ordering of the bits varies for each individual. Thus, the messy GA strings can be under-specified, exactly specified or over-specified. Missing bits of under-specified solutions are retrieved from a universal template and over-specified solutions are mapped to a string containing an appropriate number of genes. *Cut* and *splice* operators are used instead of classical crossover which allow genotypes of any length to develop over time, but in fact everything is based on an underlying fixed-length representation with a given number of loci.

The so-called Species Adaptation Genetic Algorithm (SAGA) [114] is another example for GAs which are based on variable-length representations. SAGA is inspired by the natural evolution as ordinary GAs, but additionally mimics the probably most impressive feature of evolution, i.e., that simple organisms have evolved over eons to more and more complex ones. Typically, optimizations within the SAGA framework start with relatively small genotypes which then grow with increasing number of generations. In the population individuals with different genotype sizes, the so-called *species*, coexist, and intra- as well as inter-species variation operators promote the evolutionary process until an optimum solution is found. SAGA applies classical crossover operators which break the parents at sites which are equidistant from one end of the genotype. Furthermore, some sophisticated algorithms are used which maximize the similarities between the two segments which are swapped in the one-point crossover, and also a generalization for the n-point crossover is presented in [115].

The tree-like representation of GP [34] which varies in structure and length also belongs to the class of variable-length EAs, but also the field of evolutionary electronics [116] profits from such a flexible form of representation.

In the field of structural optimization the variable-length genotype is particularly suited to perform topology optimization where an a priori unknown number of design entities needs to be found. Kim et al. [117] propose a progressive refinement process for FE-based topology optimization following the design principle "from coarse to fine". The design freedom is gradually increased by extending the genotype length in stages what leads to a reduction of computational costs for complex problems. Ryoo et al. [118] present several topology optimization problems. They apply their concept to algebraic functions, to a benchmark truss structure problem, and to a topology and layup optimization of a stiffened composite panel.

As the number of variable-length representation applications increases – only few of them are presented above – also the need for a theoretical and empirical understanding of such evolution processes grows. Kallel and Schoenauer [85] extend the *Fitness Distance Correlation* concept from Jones and Forrest [119] to the general framework of variable-length representation involving continuous parameters in order to estimate the problem difficulty and the adequacy of the representation. Ramsey et al. [120] find that the length of the individuals in the population self-adapts in direct response to the mutation rate applied. Their investigations reveal that the representation length tends to increase in the early phases of the optimization process, and then decrease to a level based on the mutation rate. Their interpretation of this phenomenon is that such algorithms have a considerable degree of self-adaptation.

5.1.2 Composite laminate optimization

Due to the ever increasing utilization of fiber-reinforced materials the need for effective optimization tools grows. A thorough literature review on the optimization of laminates and stiffened panels is presented by Venkataraman [67]. In most applications the mechanical structure is manufactured from a single laminate or the structure is subdivided into several regions with different stacking sequences. A common strategy to optimize these stacking sequences is to parameterize these laminates independently, whereas between to adjacent laminates some contiguity conditions have to be satisfied in order to ensure structural integrity (adjacent laminates must possess common plies). Such a strategy is applied in Chapter 3 where the entire rim structure is subdivided into four regions each having a maximum number of plies. For each ply the choice of material and the orientation of plies are subject to optimization and a reduction of mass can only be realized by deactivating several plies by means of boolean genes. This approach is inevitable due to the requirement for a constant-length genotype. Consequently, the size of the genotype, i.e. the size of the search space, directly depends on the maximum number of plies, although this estimation is typically too large. As soon as the number of plies is also subject to optimization, the introduction of a variable-length representations is definitely useful as will be shown in this chapter.

A different approach to composite laminate optimization is presented by Zehnder and Ermanni [121]. The so-called *patch concept* overcomes the disadvantage of invariable regions which need to be defined previous to the optimization. Instead of optimizing the stacking sequences of several laminates, an appropriate number of patches which are free to adapt their size and to move over the surface of the structural part are chosen as basic design entities. Such a patch is defined by the shape, the material, the orientation, and the position on the surface. Thus, the representation of a composite laminate can be formulated as a collection of such patches. In Zehnder and Ermanni [121] the genotype is restricted to constant length since their optimization algorithm is not suited for variable length.

The following sections combine the variable-length representation and the patch concept which is obviously well-suited for variable-length genotypes. The optimal number of patches is implicitly represented by the length of the genotype and no boolean genes activating or deactivating single patches or plies are required. Thus, the size of the search space is adapted to the problem and non-coding regions of the genotype which do not influ-

ence the appearance of the phenotype can be omitted[1]. In combination with appropriate crossover operators only exchanging entire patches, a superior optimization performance may be achieved.

Section 5.2 presents the basic requirements for a variable-length representation of composite laminates and the corresponding variation operators. Then, Section 5.3 investigates a simple eigenfrequency optimization of a rectangular plate aiming at a performance comparison of the variable-length with the traditional constant-length representation. Finally, Section 5.4 presents the optimization of the stacking sequence of a hockey stick as an example application of the variable-length representation.

5.2 Variable-length representation of composite laminates

5.2.1 The patch gene

As basic design entity a single patch is chosen which consists of several genes provided by the *UniGene* concept, see Appendix A, in order to represent the patch properties. The patch gene is defined by the position in the design space, by the orientation, by the shape, and by the material of the fiber-reinforced ply. A schematic example for a rectangle *patch gene* for a 2-dimensional panel optimization problem is shown in Figure 5.1. The shape of a patch is generally not limited, but the more complex the shape, the more defining variables need to be included in the patch gene. The genotype defining the stacking sequence of the entire panel consists of an arbitrary number of such patch genes arranged in a vector. In fact, the genotype is complex compared to a bit string representation, but the logical grouping of information is necessary to be able to define reasonable crossover operators which are exchanging homologous segments.

5.2.2 Crossover operators

The requirements for variable-length representations are stricter than for constant-length representations, where genes of different types are always at the same location within the genotype, i.e., the crossover point on one parent genotype has a distinct counterpart on the other parent genotype. For variable-length genotypes one can distinguish between intra- and inter-species crossover, i.e., crossover operations with parents of equal or unequal length, respectively. For intra-species crossover the same principles hold as

[1]Some studies have investigated the effects of non-coding regions on the recombination of building blocks in GAs [122, 123] and they found that non-coding segments reduce the disruption of building blocks. For the more complex representation and the appropriate variation operators presented in Section 5.2 this effect is assumed to be negligible.

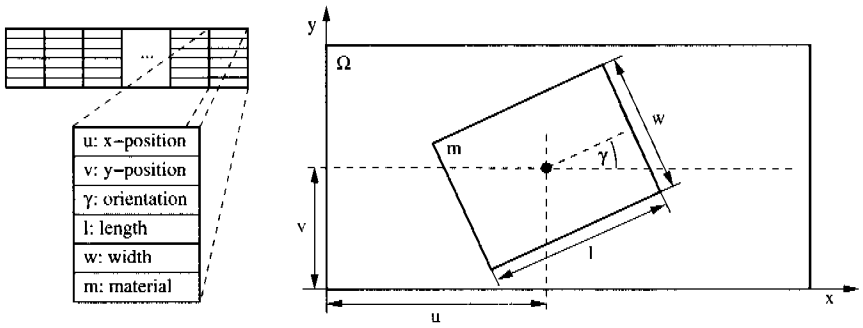


Figure 5.1: Patch gene consisting of an arbitrary number of universal gene types defining the position in the design space Ω, the size, the orientation, and the material.

for operators for constant-length genotypes, but inter-species crossover require a careful implementation. They have to satisfy two basic requirements:

- the resulting offspring need to be interpretable in the mapping stage, and
- they have to inherit meaningful 'building blocks' from their parents.

Thus, the problem for crossover operators is, given a crossover point in one parent genotype, how to identify an appropriate position to break the other parent genotype in order to exchange homologous sections.

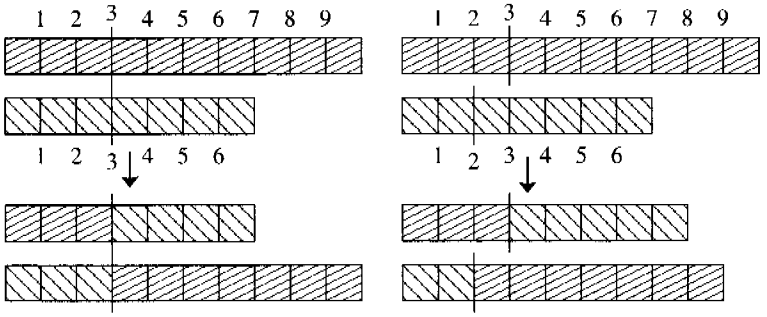
5.2.2.1 N-point crossover

The crossover operators used here are a classical and a *proportional* one-point crossover and the corresponding two-point crossovers. Since the basic design entity is a patch and the genotype consists of a list of patches, these operators are only allowed to split the genotypes between patches as illustrated in Figure 5.2. If for inter-species crossover also arbitrary crossover points within the patches would be allowed, one would run the risk of producing ill-defined offspring.

The classical one-point crossover simply determines the crossover points for both parents equidistant from one end of the potentially different-length genotypes. Consequently, this inter-species crossover only produces offspring of the same species as their parents belong to. On the other hand, the proportional crossover is able to produce offspring of different species. It requires the definition of a random variable $\lambda \in [0,1]$. The crossover points in both parents are determined according to the following formula

$$\begin{aligned} cp_1 &= \lfloor \lambda \cdot (n_1 - 1) \rfloor + 1 \\ cp_2 &= \lfloor \lambda \cdot (n_2 - 1) \rfloor + 1, \end{aligned} \tag{5.1}$$

where n_1 and n_2 are the number of patch genes of the respective parents and the numbering of the crossover points can be seen from Figure 5.2. For n-point crossovers this formula can analogously be extended by introducing a total of n random variables.



(a) classical one-point crossover

(b) proportional one-point-crossover, $\lambda = 0.3$

Figure 5.2: Classical and proportional one-point crossover with numbered crossover points.

5.2.2.2 Arithmetic crossover

The group of arithmetic crossovers as presented in Section 2.4.2 is still applicable. If the parents belong to different species, either the genes equidistant from one end of the parents can be processed, or also a proportional strategy similar to the one presented above can be chosen. The arithmetic crossovers determine new patch gene values based on the current patch gene values of both parents. Since a patch gene consists of several universal genes, the genes of the first parent are processed with the respective counterparts of the second parent.

5.2.2.3 Geometric crossover

Patches which are close to each other in the design space are not necessarily close to each other in the genotype, since the genotype is an "unordered" list of patch genes. The second requirement on crossovers mentioned above is that the offspring has to inherit meaningful 'building blocks' from their parents. This can be achieved by applying reordering algorithms or by exchanging phenotypic substructures of the parents. The latter approach is denoted as geometric crossover which simply divides the typically 2-dimensional design space into two sub-domains by placing a straight line at a random position in the design space. All the patches from both parents which are

positioned on one side of the straight line are then exchanged in order to produce two offspring individuals. Accordingly, in the 3-dimensional case the straight line is replaced by a plane.

5.2.3 Mutation operators

The classical mutation operators can be applied without any further restrictions, since they only act on a single genotype. A new category of mutation operators needs to be introduced because the traditional operators are incapable of creating genotypes of different length and species, respectively. Therefore, *add-mutation* and *delete-mutation* are introduced to change the length of genotypes. Add mutation simply adds a patch gene to the genotype which is uniformly initialized, whereas the delete-mutation removes a randomly chosen patch gene from the genotype. Needless to say, it would also be possible to add or delete a larger number of patch genes.

5.3 Simple benchmark laminate optimization

The basic motivation to investigate variable-length representation for laminate optimization is the efficiency of the optimization process. So far, the representation was of constant-length, hence, mass reduction was only possible by deactivating regions of the genotype what leads to non-coding segments.

Within this section the variable-length representation as described above is compared to a constant-length representation where each patch gene is extended with a boolean gene determining whether the patch is mapped to the phenotype or not.

5.3.1 Problem description

A concentrated mass is placed on a rectangular plate which is clamped at all edges, see Figure 5.3, and unidirectional fiber-reinforced patches have to be arranged on the plate to maximize the first eigenfrequency of the entire system. The optimization is subject to a mass constraint limiting the amount of reinforcing fiber material and regions of the plate which are not covered by at least one patch are filled with an extremely compliant fill material. The material properties are listed in Table 5.1.

5.3.2 Representation of patches

The patch genes for both the boolean and the variable-length representation are identical in terms of the genes defining

Material	UD	fill-material	mass	
E_1	135	1	1	[GPa]
E_2	10	1	1	[GPa]
ν	0.3	0.3	0.3	[-]
G_{12}	5	0.5	0.5	[GPa]
Thickness	0.15	0.15	10.0	[mm]
Area density	0.237	0.237	158	[kg/m^2]

Table 5.1: Materials of the rectangular plate optimization.

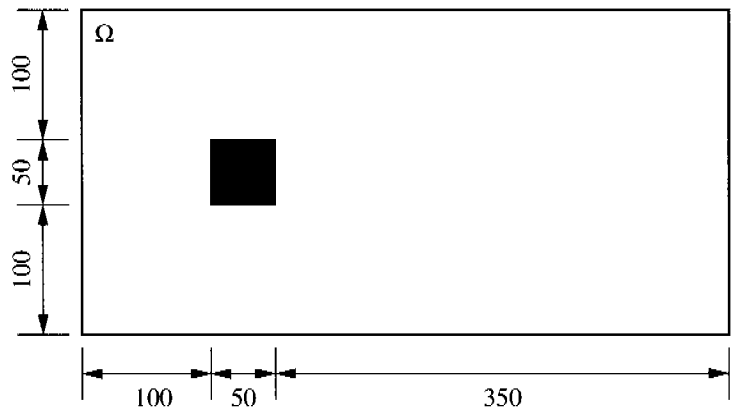


Figure 5.3: Geometrical specifications of the rectangular plate with the concentrated mass. Dimensions are given in mm.

- the position in the design space,
  u: float_gene true 250.0 true 0.0 true 500.0 50.0 50.0 false
  v: float_gene true 125.0 true 0.0 true 250.0 25.0 25.0 false
- the orientation,
  γ: *const_float_list_gene true 0.0 -90 1 90 1 12 0 30 false*
- the shape, and
  l: float_gene 1 100.0 1 50.0 1 500.0 50.0 50.0 false
  w: float_gene 1 100.0 1 50.0 1 250.0 25.0 25.0 false
- the material of the patches.
  m: string_gene true 1 0 0.0 t_ud

The only difference is that for the boolean representation an additional boolean gene (*bool_gene true true 2.0*) is added which determines whether the patch is mapped to the phenotype. It is important to note that the midpoints of the patches can be located anywhere in the design space what

leads to patches extending the design domain Ω. If this would not be permitted, the boundary regions of Ω would have a relatively lower probability of being covered.

5.3.3 FE-model

A FE-model is implemented in FELyX, see Appendix B, for the evaluation of the eigenfrequency and the mass of the entire structure. FELyX is directly integrated into the optimization framework so that the mapping from the genotype to the phenotype is extremely fast. In Figure 5.4 the mapping process is schematically depicted. The design domain is meshed with 200 finite elements and for each patch it is checked which midpoints of elements are located within the covered region. All patches represented by the genotype are processed, whereas the order of the patches in the genotype defines the stacking sequence of the laminate. The quadratic 8-node layered shell element is used allowing for the simulation of several overlapping patches, whereas the concentrated mass is also modeled as such a layer.

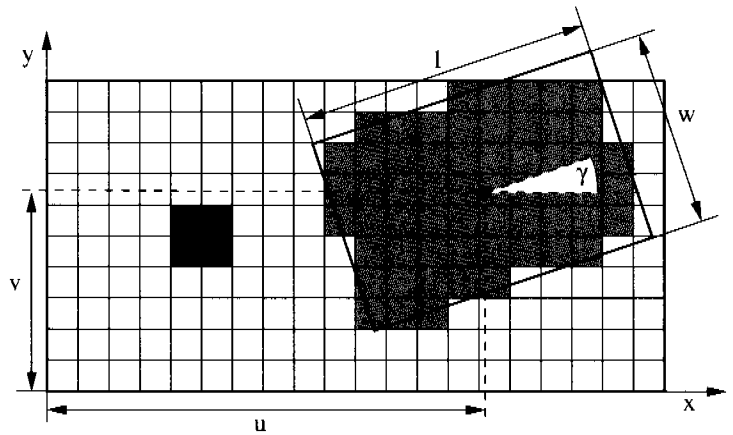


Figure 5.4: Mapping of a single patch to the FE-model.

5.3.4 Optimization setup

The boolean and the variable-length representation are compared to each other for a maximum genotype length of 20 patches. The design objective is the maximization of the first eigenfrequency subject to different maximum mass constraints. Four different setups are investigated in order to find out whether the amount of non-coding regions of the boolean representation influences the optimization performance. Therefore, the mass constraints are set to produce final solutions with approximately 4, 8, 14, and 18 patches,

respectively. For the assignment of fitness values the adaptive fitness function definition presented in Section 2.5 is employed, see Table 5.2. The population size is set to 100 individuals for all optimization setups and the total number of generations is limited to 200.

Obviously, the choice of the variation operators depends on the representation. The inter-species crossover and variable-length mutation operators are not applied to the boolean representation. Consequently, the comparability of both representation concepts is slightly limited, although the global crossover (0.8) and mutation (0.2) rates are set to equal values. Nevertheless, the comparison might indicate which of both concepts tends to provide better optimization results.

Setup	O_{init} [Hz]	O_{estim} [Hz]	C_{limit} [kg]	C_{feas_tol} [kg]
S_1	0.5	4.0	0.424625	0.02123125
S_2	0.5	8.0	0.45425	0.027125
S_3	0.5	28.0	0.5135	0.025675
S_4	0.5	50.0	0.632	0.0316

Table 5.2: Fitness definitions for setups S_1 to S_4.

5.3.5 Results

For each of the four setups a number of 30 runs are performed for the boolean as well as for the variable-length representation in order to produce statistically well-founded data. The figures 5.5 and 5.6 show the fitness and design objective convergence plots for all four setups. The median fitness value of the respective best solution of all 30 runs is plotted over the number of generations. All the convergence plots are similar, whereas the variable-length representation always outperforms the boolean representation. For the setup S_1 as well as for the setup S_2 both representation strategies converge to similar solutions. The median number of patches of the best solutions after 200 generations are equal, namely 4 for setup S_1 and 7 for setup S_2, but nevertheless the variable-length representation provides superior solutions. For all optimization setups Wilcoxon [111] rank-sum tests are conducted and the results indicate a 99% confidence level that the variable-length representation finds superior solutions after 200 generations.

The purpose of this investigation is to find out whether the reduction of non-coding regions leads to a better optimization performance. When comparing the setup S_1 to the setup S_4 which has significantly less non-coding regions for the boolean representation, this hypothesis needs to be partially rejected - at least for the optimization example at hand. The convergence plots of these setups are similar, i.e., the divergence of the fitness plots of setup S_1 is not significantly larger compared to setup S_4 and

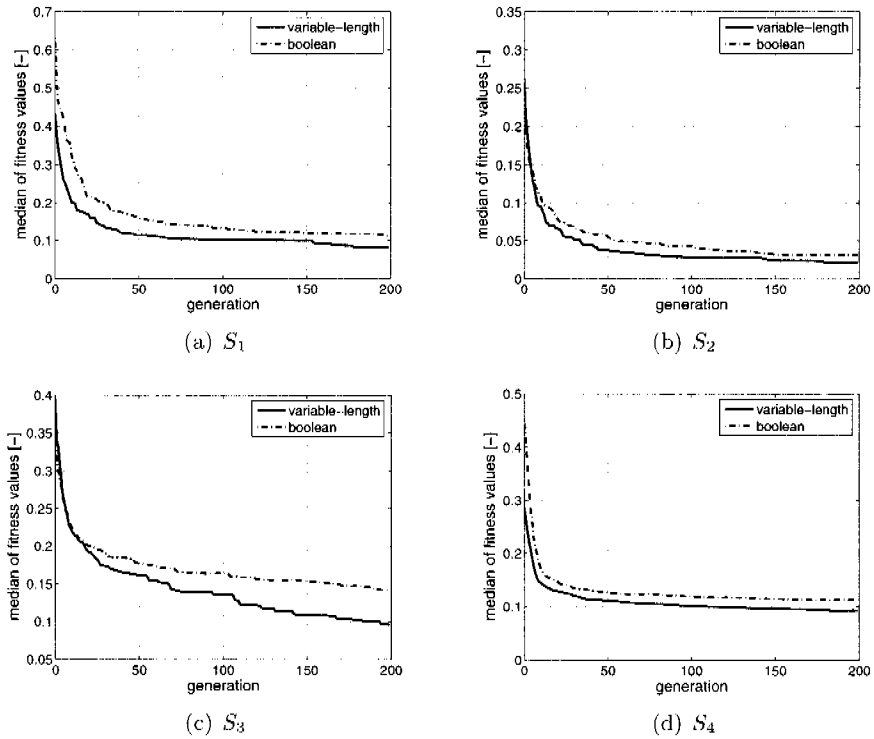
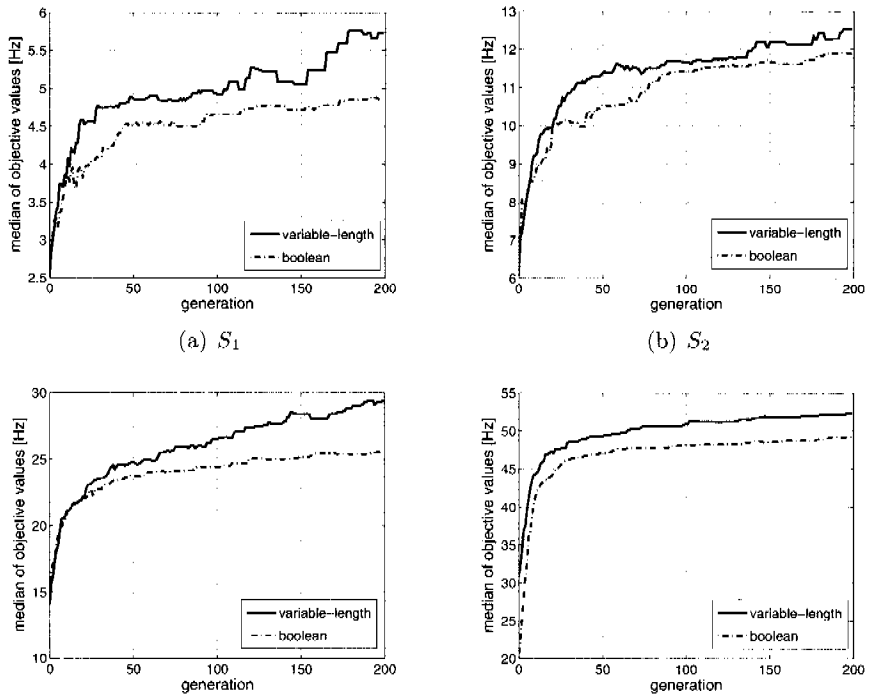


Figure 5.5: Median fitness values from the best solutions of 30 optimization runs for boolean and variable-length representation.

also the setup S_2 shows an analogous behavior. The only exception is setup S_3, where the fitness plots show a slightly more divergent behavior.

The amount of non-coding regions seems not to be the only explanation for the superior convergence behavior of the variable-length representation. A possible explanation is the basically larger search space for the boolean representation, since each patch gene consists of an additional boolean gene. A further major difference between the boolean and the variable-length representation is the application of different variation operators, i.e., inter-species crossover and variable-length mutation are applied for the latter representation concept. Due to the low operator rates of add- and delete-mutation their effect is supposed to be almost negligible. On the other hand, the inter-species crossover, in particular the proportional one- and two-point crossovers, are applied with relatively high operator rates. Consequently, their influence on the optimization performance could be a reason for the differences in efficiency.

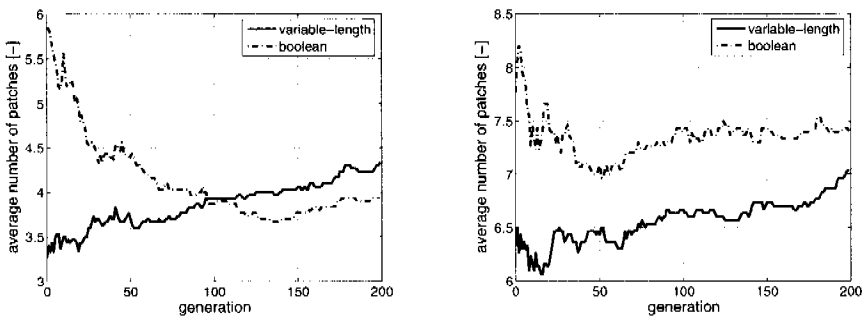


(a) S_1 (b) S_2

(c) S_3 (d) S_4

Figure 5.6: Median objective values from the best solutions of 30 optimization runs for boolean and variable-length representation.

A potential problem known from GP applications is the occurrence of so-called *bloats*. Such bloats describe the tendency of the solutions to grow in length the longer the optimization process runs. In particular for the setups S_1 and S_2 this phenomenon would have to occur due to their low mass constraint. In Figure 5.7 the respective average number of patches are shown. Although the average number of patches of the variable-length representation are slightly increased during the optimization process this cannot be explained with the bloat phenomenon. It is rather the continuing optimization process finding improved solutions with smaller patches which locally increase the stiffness of the structure and therefore increase the first eigenfrequency. In the beginning of the optimization process it is much more likely that individuals with few relatively large patches are superior, whereas the optimization process is typically able to arrange more but smaller patches later on. It is in the nature of the variable-length patch representation that no non-coding sequences in the genotype can occur, i.e.,



(a) S_1 (b) S_2

Figure 5.7: Median number of patches from the best solutions of 30 optimization runs for boolean and variable-length representation.

each patch of a genotype is mapped to the FE-model what automatically leads to an increased mass of the structure. Additionally, the minimum patch size is given by the gene definitions for the length and the width of the patch and therefore the number of patches cannot grow unlimited.

The results gained from this optimization example cannot be generalized to any other variable-length optimization concept, but at least for composite laminate optimization tasks the presented approach is promising. The next section presents the application of the variable-length laminate optimization concept to a problem from engineering practice the optimization of a hockey stick.

5.4 Hockey stick optimization

The variable-length representation concept for composite laminates was applied to the optimization of a hockey stick [124]. The main goal of this application is to show the applicability of the variable-length representation concept to arbitrary laminate optimization problems, hence, no performance and solution quality comparison to a fixed-length representation concept is carried out. The design objective is to determine a stacking sequence with minimal mass in consideration of stiffness and cost constraints. Only the basic idea and results are presented within this section in order to demonstrate the capabilities of the laminate optimization framework. A presentation of all results would go far beyond the scope of this chapter.

5.4.1 FE-model

The hockey stick is manufactured from a foam core which is surrounded by carbon, glass, and/or aramid fiber-reinforced plastics. The shaft and the blade have to fulfill several requirements in terms of stiffness and strength, hence, the stacking sequence needs to be varied in different regions of the stick.

For the construction of a hockey stick two different load cases have to be considered. First, a lateral load case as shown in Figure 5.8 allows for an estimation of the stiffness properties of the stick. A second load case models the ultimate load applied when a slap shot occurs, see Figure 5.9. For both load cases a maximum deflection can be measured in the middle of the shaft and at the blade, respectively.

The FE-model is built from quadratic 8-node layered shell elements which model the reinforcement plies and the foam core is modeled by quadratic 10-node tetrahedral solid elements.

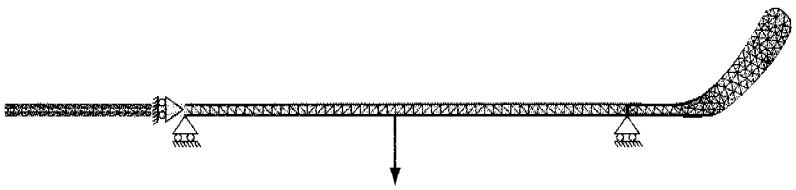


Figure 5.8: Lateral displacement load case.

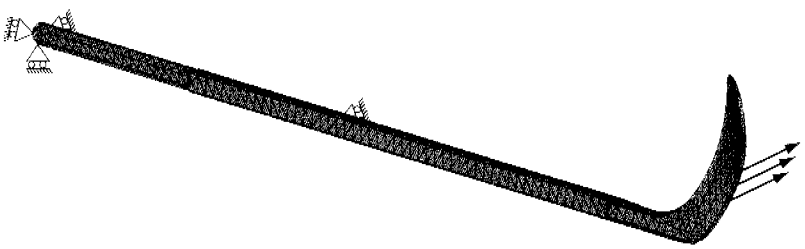


Figure 5.9: Shot load case.

5.4.2 Representation

Obviously, the patch gene as it is introduced in Section 5.2 cannot be transfered to this optimization in its original form. The basic shape of a patch

is still rectangular, but the mapping from genotype to phenotype is not 2-dimensional anymore. Consequently, the patch representation needs to be slightly adapted to the new design space, i.e. the surface of the hockey stick. In Figure 5.10 a single patch is depicted with its defining parameters. The positioning of the patch is determined by the distance x between the midpoint of the patch and the shaft end. The length l of the patch is defined in the direction of the shaft and the orientation γ is given relative to this direction. An adaptation is required for the definition of the patch geometry along the circumference of the shaft. From a manufacturing point of view it is not reasonable to use patches which start or end in the middle of a shaft surface. Thus, the starting edge of the patch is defined by a further parameter which can take values from 1 to 4 as indicated in Figure 5.10. Another parameter which can also take values from 1 to 4 is finally introduced defining how many shaft surfaces are covered by the respective patch, i.e., a value of 1 produces a patch which covers only one surface, a value of 4 leads to a patch covering the entire circumference. Finally, a material parameter defines which fabric is chosen for the respective patch.

A base laminate is used for the entire shaft and the reinforcing patches are exclusively arranged on the shaft. The blade has a predetermined stacking sequence which is not subject to optimization. The mapping from genotype to phenotype is implemented by incorporating the FELyX library into the optimization framework. Thus, each finite element can be directly accessed and the respective patches can be mapped to the FE-model.

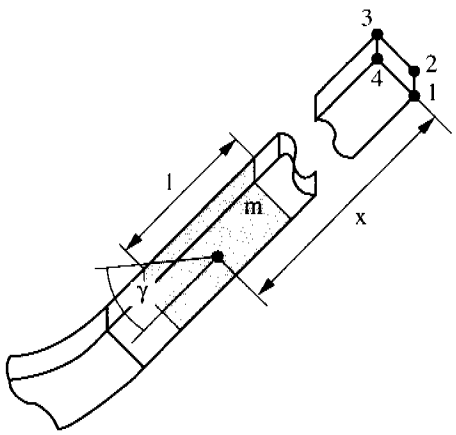


Figure 5.10: Hockey stick patch gene.

5.4.3 Optimization formulation

The main goal of this optimization is to determine a stacking sequence possessing the same structural stiffness as state-of-the-art hockey sticks, but the mass should be reduced as much as possible. Furthermore, a cost constraint has to be satisfied in order to comply with a given maximum production cost. Unfortunately, failure criteria could not be considered within the scope of this work, since FELyX did not provide the required postprocessing routines at this time.

The fitness formulation introduced in Section 2.5 is employed for this optimization. The mass of the stick is the design objective and three additional constraints are to be considered. The costs are formulated as an upper limit constraint, whereas both the lateral and the shot deflection are defined as target constraints.

	Mass	Costs	Lateral deflection	Shot deflection
O_{init}	0.35 [kg]			
O_{estim}	0.18 [kg]			
C_{limit}		28 [Fr]		
C_{target}			10 [mm]	450 [mm]
C_{feas_tol}		2 [Fr]	5 [mm]	250 [mm]
C_{adm_tol}			0.15 [mm]	10 [mm]

Table 5.3: Objective and constraint definitions for the hockey stick optimization.

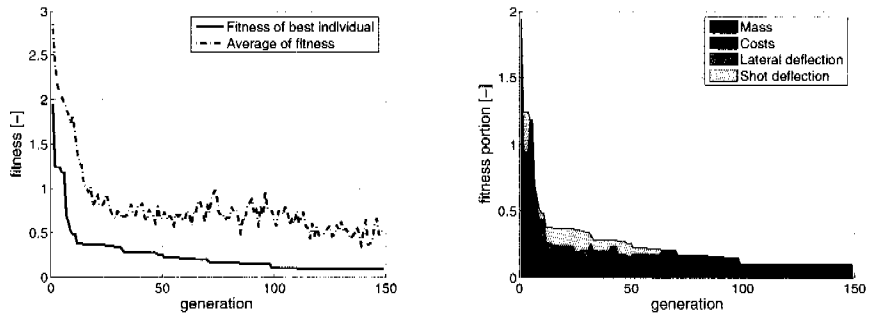
Four different materials can be used for the laminate, namely carbon- and glass-fiber reinforced unidirectional plies, and carbon- and glass-fiber reinforced woven fabrics. The properties of these standard fabrics are not listed within this thesis, but they can be found in [124].

The population size is defined to be 56 and the same variation operators as presented above are employed. The geometric crossover is appropriately adapted in order to exchange patches from either the lower or the upper part of the shaft.

5.4.4 Results

The optimization is completely random initialized and the maximum number of patches is set to 12. Since already a base laminate is applied and only the reinforcement plies are subject to optimization, this number is chosen rather high. The convergence behavior of the optimization run is plotted in Figure 5.11. During the optimization the number of patches is continuously reduced until the best solutions consists of only two large patches covering almost the entire shaft. The mass can be clearly reduced and the cost

constraint is never violated, whereas both target constraints are satisfied after 150 generations. However, the results have to be interpreted carefully, since the missing failure criteria should be considered as well. Unfortunately, it cannot be guaranteed that in some highly stressed regions the failure criteria could be violated, but in this case, the optimization algorithm should be able to locally reinforce the structure with an additional small patch.



(a) Fitness of best individual and average fitness of population

(b) Fitness portions of best individual

Figure 5.11: Result plots of the hockey stick optimization.

5.5 Conclusion

Within the scope of this chapter a variable-length representation concept for composite laminate optimization is presented. It is a combination of variable-length representation, which is widely used in the evolutionary optimization community, and the patch concept for composite laminate optimization.

A benchmark problem dealing with the maximization of the first eigenfrequency of a rectangular plate indicates that the variable-length approach, at least in its current implementation, performs superior to the boolean representation approach which has been used so far for composite laminate optimization. Furthermore, the optimization of a hockey stick demonstrates the applicability of the concept to problems from engineering practice, although the simulation model used lacks the consideration of failure criteria. Just recently FELyX has been provided with the postprocessing of state-of-the-art failure criteria so that this aspect could also be considered in future laminate optimization problems.

A possible drawback of the presented method is the mapping stage where

the rectangular patch is mapped to the FE-model. First of all, the patch should not be necessarily of rectangular shape, i.e. other shapes should also be possible in order to only place fiber fabrics where they are really required. This could be achieved by introducing patches of more complex shape, but the search space would grow accordingly. A second issue is the injectivity of the mapping process. The patches can cross the boundaries of the design space and the rectangular patches are discretized depending on the granularity of the FE-mesh, hence, different genotypes can be mapped to identical phenotypes what may affect the search process adversely. In Chapter 7 a novel approach to laminate optimization is presented which is based on mathematical graphs as representation concept addressing the identified drawbacks of this method.

Chapter 6

Graph-based truss optimization

This chapter is based on the publication: *Evolutionary truss topology optimization using a graph-based parameterization concept* [92]

6.1 Introduction

In the field of truss optimization a variety of methods have been developed in the last decades. A lot of research has been done on the ground structure approach initiated by Dorn [55] where the members and the nodes are selected from a highly connected ground structure. The cross-sectional areas of the members are considered as continuous design variables, whereas the nodal locations are fixed [125] or can be moved [126, 127] in order to minimize the mass or the compliance of the truss structure. Most often the optimization objective is restricted to comply with stress and eigenfrequency constraints as well as local and global stability requirements as for example can be seen in Pedersen and Nielson [128]. These types of constraints complicate the search for the global optimum due to the phenomenon of singular topologies. A thorough summary of many methods dealing with design-dependent constraints, e.g. the ϵ-relaxation approach [129], can be found in Rozvany [130]. Furthermore, stress ratio and compliance based methods, e.g. Fully Stressed Design (FSD) or Uniform Energy Distribution (UED), have been thoroughly investigated. A critical review of these and further popular methods, e.g. Evolutionary Structural Optimization (ESO, see [131]) or Adaptive Biological Growth (ABG), can be found in [132].

Besides all the above mentioned methods, the application of GA to truss optimization has been investigated in several publications. Hajela and Lee [56] use GA to develop near-optimal topologies by subdividing the optimization task into two stages. The first stage generates a number of kinematically stable truss topologies, whereas the second stage is dedicated

to the member sizing in order to get a minimum weight structure. Azid et al. [59] employ evolutionary genetic search in combination with a slightly modified ground structure approach for the layout optimization of a three-dimensional truss and Lingyun et al. [133] successfully apply a niche hybrid GA to truss optimization with frequency constraints. Kawamura et al. [134] present an interesting representation approach by using triangular elements as basic design entity. The topologies produced by this method are guaranteed to have neither needless members nor undesirable overlaps between members and only stable structures can occur. All these examples show that GAs are also well-suited for the optimization of truss structures, although a global optimum solution can hardly be found. Furthermore, these examples have in common that they are based on binary or real-valued representation, i.e., the design variables are organized in vectors which are manipulated by variation operators according to the basic theory of EA.

Basically, the topology of a truss structure can also be considered and modeled as a mathematical graph (cf. Appendix C). The nodes of the truss structure can be regarded as vertices, whereas the members correspond to the edges of the graph. Kawamoto et al. [135] use the graph representation in combination with the ground structure approach to perform topology optimization of symmetric mechanisms. Their idea is to embed some topological graphs into the ground structure in order to reduce the complexity of the optimization problem.

In this chapter a novel combination of EAs and mathematical graph theory is applied to truss optimization. The complete truss topology is represented by a graph, whereas each vertex corresponds to a node and each member is described by an edge. In other words, instead of holding the information of the truss structure in a conventional 1-dimensional genotype, the graph itself is considered as the genotype and special variation operators are directly applied on it. Additionally, some typical graph algorithms, e.g. Cuthill-McKee reordering and connectivity analysis, are used in order to improve the optimization efficiency or filter out globally unstable solutions.

The main motivation to investigate this combination of methods is the independence of any kind of ground structure. Furthermore, this approach overcomes the ordinary restriction of having constant-length genotypes for genetic search, i.e., the number of design entities must remain unchanged where the only way to remove members from the truss structure is to let the cross-sectional area be null or to use a binary gene determining whether the member exists or not. Only Ryoo and Hajela [118] investigate a binary-encoded representation concept based on variable-length genotypes. The graph representation approach provides more flexibility, since the size of the graphs can arbitrarily change during the optimization process and even graphs of unequal size can be recombined.

In Section 6.2 the graph genotype as representation concept is defined. Section 6.3 deals with graph variation operators required for mutating and

mating graph individuals. Finally, Section 6.4 presents three planar numerical examples in order to demonstrate the strength of the concept at hand. Moreover, an introduction to graph theory explaining the basic notions is given in Appendix C.

6.2 The graph genotype

6.2.1 Representation requirements

In Figure 6.1 a classic two-dimensional truss structure is depicted with its corresponding graph genotype. For each node of the structure three different data types need to be known:

- a unique identifier,
- the planar or spatial coordinates,
- and a boolean parameter determining whether the node can be moved during the optimization process or not. Clamped or loaded nodes must not be relocated.

Additionally, some information about each member is required:

- the two unique identifiers of the member's nodes,
- and the cross-sectional area.

All the above mentioned information is required to unambiguously define the truss structure and has therefore to be represented by the graph genotype.

According to Balakrishnan and Ranganathan [136] a *simple* graph has no loops and no multiple, i.e. parallel, edges and in computer science such

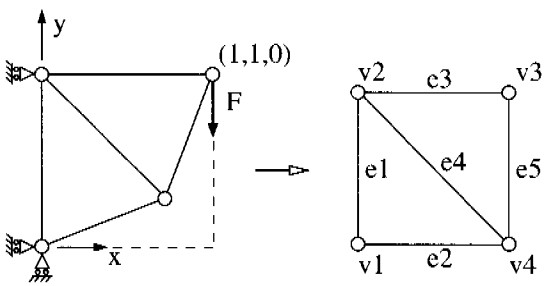


Figure 6.1: Simple truss structure and the schematic illustration of its corresponding graph genotype.

vertex	label	x	y	movable
$v1$	1	0	0	false
$v2$	2	0	1	false
$v3$	3	1	1	false
$v4$	4	[0,1]	[0,1]	true
edge	label 1	label 2	weight/area	
$e1$	1	2	1	
$e2$	1	4	1	
$e3$	2	3	1	
$e4$	2	4	1	
$e5$	3	4	1	

Table 6.1: Node/vertex and member/edge properties.

graphs are generally termed *acyclic*. Loops do not make sense in a truss structure as they only contribute to the overall mass but not to the structural stiffness. Parallel edges are undesired because they would unnecessarily increase the complexity of the structure and they could easily be replaced by single edges. Furthermore, the requirement of a unique identifier for each node can be realized by using a labeled graph, whereas it is reasonable to choose integers as labels. The coordinates of the nodes and the boolean parameter are assigned to the graph genotype's vertices by introducing ordinary vertex properties. As stated in Definition 11 in Appendix C.1 each edge is defined by an unordered pair of incident vertices. In practice, these vertices are identified by their labels, hence, each edge can be represented as a pair of labels. Finally, the cross-sectional area of each member can easily be mapped to the graph genotype by interpreting it as an edge weight.

Summarizing, the graph genotype needs to be a simple, labeled, and weighted graph supplemented with additional vertex properties. Each vertex holds properties defining its label (unique integer), the position (three floating-point numbers for the spatial coordinates) and whether it is allowed to move or not (boolean variable), whereas every edge holds the information about its endpoints (labels of respective nodes) and its cross-sectional area.

Recalling the example illustrated in Figure 6.1, the Table 6.1 lists all required information. The members should all have a cross-sectional area of unity. Only vertex $v4$ is neither clamped nor loaded, thus, its position can be anywhere in the square design space.

Rozvany [83] defines *layout optimization* as the simultaneous selection of the optimal *topology* (spatial sequence or connectivity of members), *geometry* (location of intersections), and *cross-sectional dimensions* (sizing). The graph genotype allows for the concurrent optimization of all these aspects of structural optimization, although this leads to a tremendous complexity

of the optimization task. The topology of a truss structure can be modified by either changing label 1 or label 2 of an edge (or both) or adding and removing members. The coordinates of the nodes define the geometry of the truss structure. It is important to strictly distinguish between clamped or loaded nodes, they must not be moved, and free nodes which can be placed anywhere in the design space. Consequently, only the coordinates of movable nodes are subject to changes during the optimization process. Finally, the sizing optimization is realized by modifying the edge weights of the respective members. Needless to say, separate optimization of the above-mentioned properties is possible, but not recommended, since a separately optimized topology may no longer be optimal if the geometry is changed.

6.2.2 Graph genotype analysis

Each graph genotype produced during the optimization process has to be mapped to a FE-model for the evaluation of its fitness composed of objective and constraint values. Unfortunately, not each graph genotype (or individual when speaking in terms of evolutionary optimization) can be successfully mapped to a running FE model. After applying variation operators individuals can occur representing novel truss topologies which would fail in the evaluation step. Thus, there are several obvious preconditions a graph genotype has to fulfill in order to guarantee feasibility of the truss structure:

1. no non-connected subgraphs[1] may occur,
2. all the clamped and loaded vertices (nodes) must be connected, and
3. the represented truss topology must be statically determinate.

It would be absolutely useless to start the mapping and evaluation process for individuals not complying with these requirements, hence, the optimization efficiency would be decisively decreased. Such individuals as for example depicted in Figure 6.2 need to be filtered out before by applying graph analysis algorithms.

Connected components A *path* is a sequence of vertices where each vertex is connected by an edge to the subsequent vertex in the path. If there exists a path from vertex u to v, then vertex v is said to be *reachable* from u. A *connected component* is a group of vertices in an undirected graph which are all reachable from one another, whereas a single vertex is considered to form the smallest possible component. The connected-components algorithm (based on the depth-first search algorithm, for details see [137]) is able to quickly determine such connected components. Not only the total number of connected components is given, but also the grouping

[1] A graph H is called a *subgraph* of G, if $V(H) \subseteq V(G)$

of the vertices is provided. Thus, the connected-components algorithm is perfectly suited to check the first and the second precondition.

In order to comply with the first precondition, the number of connected components must either exactly be equal to one, i.e. all vertices are connected to each other, or, if there are more connected components, only one of them may contain more than one vertex. Furthermore, the second precondition says that no clamped or loaded nodes may be represented by isolated vertices. Consequently, all the corresponding vertices must be contained in the same connected component.

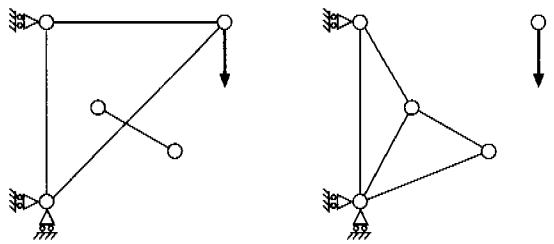


Figure 6.2: Two undesired topologies filtered out due to violation of precondition 1 (left) and 2 (right), respectively.

Statical determinacy The statical determinacy of a truss structure can be estimated by applying Maxwell's algebraic rule (see e.g. Fowler and Guest [138] and references therein). Generally, the rule can be formulated as

$$f = d \cdot n(G(V,E)) - m(G(V,E)) - s, \tag{6.1}$$

where d is the dimensionality, s is the number of supports, and n and m are the order and the size (cf. Appendix C) of the mathematical graph representing the truss structure, respectively. If $f = 0$, at least a necessary but not in general sufficient condition for establishing statical determinacy is fulfilled. Individuals for which it holds $f < 0$ are statically overdetermined and can also be considered as feasible solutions. However, each individual is checked for compliance with Equation 6.1 and the FE-evaluation of individuals not fulfilling the condition is omitted.

Nevertheless, all these individuals violating at least one precondition are consciously kept in the optimization process and the population, respectively, since their 'genetic material' is not necessarily useless. It would be a possibility to repair individuals not complying with the preconditions by applying further graph algorithms and inserting edges in order to stabilize the truss structures, but practice has shown that in an optimization run only a low percentage of individuals are infeasible.

6.2.3 Implementation

The entire code is implemented in C++ and based on four libraries. As optimization engine, the Evolving Objects library [93] (EOlib) is employed providing the basic functionalities required for evolutionary computation. The implementation of the graph genotype itself is a combination of the Boost Graph Library [137, 139] (BGL) and the universal genotype concept (cf. Appendix A). The vertex labels are represented by integer-genes providing lower and upper limits, the vertex coordinates and the edge weights are described by float-genes, and a bool-gene indicates whether a vertex is movable or not.

Due to the huge number of evaluations typically required for evolutionary optimizations, FELyX (cf. Appendix B) is utilized allowing for extremely fast mapping and evaluation procedures. The graph genotype properties are mapped to 3D-spar uniaxial tension-compression elements with three degrees of freedom at each node. For any computation the self-weight of the truss members is not taken into consideration. The build-up of the FE model is completed by inserting the boundary conditions of clamped and loaded nodes. Finally, parallel computing is made possible by including the Parallel Virtual Machine [99] (PVM) library.

6.3 Graph variation operators

It is crucial that the variation operators cover all aspects of layout optimization, i.e., they have to allow for changes concerning the topology, the geometry, and the sizing of the truss structures. There are a lot of possible operators acting on vertices and edges and their properties, respectively. The only restriction is to keep the number of vertices (connected or unconnected) constant, otherwise some graph algorithms cannot be applied anymore. For the illustration of the effect of mutation and crossover operators the *adjacency matrix* is used which is defined in Appendix C.2.

6.3.1 Mutation operators

Generally, topology mutation can be considered as a transformation $\mathcal{M}$: $A_o \rightarrow A_{mut}$ of the respective graph adjacency matrix, whereas the geometry and the sizing can be mutated by altering the respective vertex and edge properties.

6.3.1.1 Topology mutation operators

Random removal or insertion of edges Probably the simplest possible mutation operator concerning topology optimization is the random insertion or removal of edges. A random number generator (RNG) is used to determine an edge to be removed. Analogously, two vertices are chosen by RNG

which are newly connected with an edge, whereas the edge weight, i.e. the cross-sectional area of the newly created member, is also determined by chance. In the example below the edge $(v2, v3)$ is removed while another edge $(v1, v3)$ is inserted.

$$A_o = \begin{bmatrix} 0 & w_{12} & 0 & w_{14} \\ w_{12} & 0 & w_{23} & w_{24} \\ 0 & w_{23} & 0 & w_{34} \\ w_{14} & w_{24} & w_{34} & 0 \end{bmatrix} \rightarrow A_{mut} = \begin{bmatrix} 0 & w_{12} & w_{13} & w_{14} \\ w_{12} & 0 & 0 & w_{24} \\ w_{13} & 0 & 0 & w_{34} \\ w_{14} & w_{24} & w_{34} & 0 \end{bmatrix}$$

Removal or connection of free edges It might happen that individuals occur having vertices with only one incident edge. Obviously, such edges are not reasonable and do not contribute to an optimum truss structure. Consequently, such vertices need to be completely isolated (if they are not loaded or clamped) or a second edge (or also a third edge in the three-dimensional case) needs to be inserted in order to stabilize the truss structure. Newly inserted edges are initialized with random edge weights.

Flipping edges This operator changes the topology by flipping edges, i.e., one endpoint of an existing edge is reconnected to another vertex, whereas the new endpoint is determined by chance. The edge weight of flipped edges is not changed as can be seen in the example below

$$A_o = \begin{bmatrix} 0 & w_{12} & 0 & w_{14} \\ w_{12} & 0 & w_{23} & w_{24} \\ 0 & w_{23} & 0 & w_{34} \\ w_{14} & w_{24} & w_{34} & 0 \end{bmatrix} \rightarrow A_{mut} = \begin{bmatrix} 0 & w_{12} & w_{13} & w_{14} \\ w_{12} & 0 & w_{23} & w_{24} \\ w_{13} & w_{23} & 0 & 0 \\ w_{14} & w_{24} & 0 & 0 \end{bmatrix},$$

where $w_{13} = w_{34}$.

Merging or disconnection of vertices A general problem when applying variation operators is that the structure can hardly become simpler with ordinary removal of edges. If a truss structure is statically determinate one cannot simplify it by removing a single edge, since this transforms the truss topology to a mechanism. Therefore, two similar strategies are implemented as mutation operators. Either a vertex is completely disconnected from the remaining structure, i.e., all the edge weights in one row and its corresponding column in the adjacency matrix are deleted, or the incident edges of two vertices are merged as can be seen in the example below ($v3$ and $v4$ are merged)

$$A_o = \begin{bmatrix} 0 & w_{12} & 0 & w_{14} \\ w_{12} & 0 & w_{23} & w_{24} \\ 0 & w_{23} & 0 & w_{34} \\ w_{14} & w_{24} & w_{34} & 0 \end{bmatrix} \rightarrow A_{mut} = \begin{bmatrix} 0 & w_{12} & w_{13} & 0 \\ w_{12} & 0 & w_{23} & 0 \\ w_{13} & w_{23} & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

where $w_{13} = w_{14}$. If both involved vertices are movable, the new vertex lies exactly in between the two original vertices.

6.3.1.2 Geometry mutation operators

The geometry of the truss structure can only be influenced by the vertex locations. Thus, mutating the vertex locations leads to a direct geometric variation of the original individual. There are two types of mutation, i.e. the uniform and the Gaussian mutation strategies, which are applied to all vertex location coordinates with a certain probability. Since the vertex locations are implemented by float-genes, these mutations can be applied in a straightforward way.

6.3.1.3 Sizing mutation operators

The sizing mutation operators work similar to the geometry mutation operators. Analogously, an uniform and a Gaussian mutation strategy is implemented for mutating the edge weights of the truss structure.

6.3.2 Crossover operators

The crossover operators can be considered as transformations $C : A_{p1}, A_{p2} \rightarrow A_{o1}, A_{o2}$ of two parent to two offspring individuals. They work most efficiently if they exchange substructures of the original trusses and therefore the Cuthill-McKee reordering algorithm is employed (cf. Appendix C.3). The purpose of applying Cuthill-McKee reordering is to precondition the graphs representing truss structures in terms of similarity. In order to achieve maximum similarity the starting vertices of the reordering algorithm of both original graphs have to be the same, i.e. they should have the same coordinates. It is therefore most reasonable to choose a non-movable vertex (clamped or loaded) at the boundary of the design space as starting vertex. A similar vertex labeling of the original graphs guarantees that the truss structures can be split at similar positions, hence, more realistic structures occur after the crossover operations.

6.3.2.1 Topology crossover operators

Proportional edge crossover This crossover type is similar to an ordinary one-point crossover for traditional 1-dimensional vector genotypes, but it also works for graphs of unequal edge numbers ($m(G_{p1}) \neq m(G_{p2})$). After Cuthill-McKee reordering of both parent individuals and determining a random number $r \in [0, 1]$, the number of edges of the parent graph genotypes A_{p1} and A_{p2} are multiplied by r and rounded to integer numbers to get the split positions in the adjacency matrices. Two offspring graph genotypes A_{o1} and A_{o2} are then created by exchanging all edges following after the

respective split positions as can be seen in the example below. The value of r is set to 0.52 and the edge weights of parent A_{p1} are equal unity and the edge weights of A_{p2} are equal to two for simpler identification of parents and their offspring. The subscripts indicate the number of the respective edge.

$$A_{p1} = \begin{bmatrix} 0 & 1_1 & 1_2 & 1_3 & & \\ 1_1 & 0 & 1_4 & & & \\ 1_2 & 1_4 & 0 & 1_5 & 1_6 & \\ 1_3 & & 1_5 & 0 & 1_7 & \\ & & 1_6 & 1_7 & 0 & \\ & & & & & 0 \end{bmatrix} \quad A_{p2} = \begin{bmatrix} 0 & 2_1 & 2_2 & & & \\ 2_1 & 0 & 2_3 & 2_4 & & \\ 2_2 & 2_3 & 0 & 2_5 & 2_6 & 2_7 \\ & 2_4 & 2_5 & 0 & 2_8 & 2_9 \\ & & 2_6 & 2_8 & 0 & 2_{10} \\ & & 2_7 & 2_9 & 2_{10} & 0 \end{bmatrix}$$

The split position for the first parent is after the fourth edge and the second parent is split after the fifth edge. Consequently, this leads to offspring of the following topology.

$$A_{o1} = \begin{bmatrix} 0 & 1_1 & 1_2 & 1_3 & & \\ 1_1 & 0 & 1_4 & & & \\ 1_2 & 1_4 & 0 & & 2_6 & 2_7 \\ 1_3 & & & 0 & 2_8 & 2_9 \\ & & 2_6 & 2_8 & 0 & 2_{10} \\ & & 2_7 & 2_9 & 2_{10} & 0 \end{bmatrix} \quad A_{o2} = \begin{bmatrix} 0 & 2_1 & 2_2 & & & \\ 2_1 & 0 & 2_3 & 2_4 & & \\ 2_2 & 2_3 & 0 & 2_5 & 1_6 & \\ & 2_4 & 2_5 & 0 & 1_7 & \\ & & 1_6 & 1_7 & 0 & \\ & & & & & 0 \end{bmatrix}$$

In Figure 6.3 the parents and their offspring are depicted.

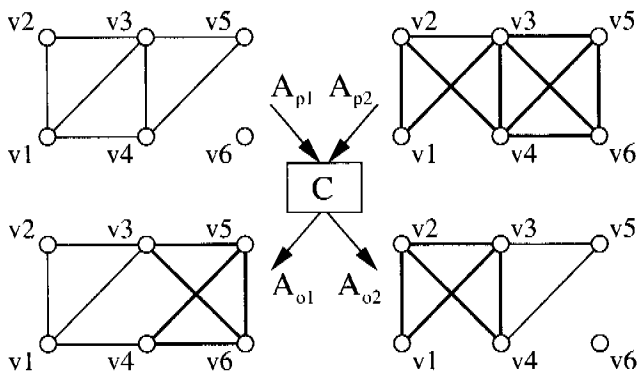


Figure 6.3: Parents and offspring of a proportional edge crossover.

Uniform edge exchange The second topology variation operator exchanges incident edges of some randomly chosen vertices. After the Cuthill-McKee reordering is executed an equally labeled vertex is chosen from each

parent genotype. The respective rows and columns of these vertices are then simply exchanged leading to topologically novel offspring individuals. The edge weights of all incident edges of both vertices remain unchanged. As an example, the incident edges of vertices $v6$ of the parent individuals A_{p1} and A_{p2} in Figure 6.3 are exchanged. Below the resulting adjacency matrices of the offspring are shown.

$$A_{o1} = \begin{bmatrix} 0 & 1_1 & 1_2 & 1_3 & & \\ 1_1 & 0 & 1 & & & \\ 1_2 & 1_4 & 0 & 1_4 & 1_5 & 2_7 \\ 1_3 & & 1_4 & 0 & 1 & 2_9 \\ & & 1_5 & 1 & 0 & 2_{10} \\ & & 2_7 & 2_9 & 2_{10} & 0 \end{bmatrix} \quad A_{o2} = \begin{bmatrix} 0 & 2_1 & 2_2 & & & \\ 2_1 & 0 & 2_3 & 2_4 & & \\ 2_2 & 2_3 & 0 & 2_5 & 2_6 & \\ & 2_4 & 2_5 & 0 & 2_8 & \\ & & 2_6 & 2_8 & 0 & \\ & & & & & 0 \end{bmatrix}$$

6.3.2.2 Geometry crossover operators

The aim of crossovers altering the geometry of some individuals is to average two parent individuals regarding their vertex coordinates. In other words, after the Curhill-McKee reordering is executed the operator iterates over all vertices vi of both parent individuals and geometrically mates them with a certain probability. The identically labeled vertices of both parents are processed with hypercube and segment crossovers as they are introduced in Section 2.4. Provided that their vertex coordinates are allowed to be changed, each component of the vertex coordinates is modified.

6.3.2.3 Sizing crossover operators

The sizing crossover operators work similar to the geometry crossover operators. First, the Cuthill-McKee reordering is also applied for the preconditioning of the parent individuals. Analogously to the geometry crossovers, segment and hypercube crossover strategies are implemented for mating the edge weights of the parent individuals.

6.4 Numerical examples

Three numerical examples are presented in this section. The first example is a variation of the ten-bar truss problem known from several publications [125, 140, 141] which is examined for validation purposes. The second example is similar, but the size of the structure is increased and it illustrates the independency from any kind of ground structure. Finally, the third example discusses the optimization of a 52-bar planar truss known from several publications [142, 143, 144]. The full capacity of the graph representation is demonstrated by reformulating this example in a much freer way allowing for arbitrary topologies and geometries of the structure. For

all presented truss structures circular cross sections are used, but the concept is absolutely not limited to such simple geometries. These examples are chosen to demonstrate the applicability of the graph representation to simple examples, but it is not intended to compete with other algorithms in terms of efficiency yet.

6.4.1 The modified ten-bar truss optimization

Most often, the ten members of the ground structure indicated by solid lines in Figure 6.4 are optimized by MP methods. However, this example can also be treated by using the graph genotype concept. In a first step the positions of the nodes are fixed and only the optimum number of members m and their sizes $\mathbf{a}^T = [a_1, a_2, \ldots, a_m]$, i.e. their cross-sectional areas, should be optimized. In contrast to the traditional problem formulation, all of the nodes may be connected to each other, hence, a total number of $m_{max} = n \cdot (n-1)/2 = 15$ members is possible, where $n = 6$ is the total number of nodes. A minimum number of $m_{min} = 4$ is required in order to form the simplest possible statically determinant truss topology. Since the optimum number of members is not known in advance, the graph genotype is allowed to contain between four and fifteen edges. The formulation of the optimization problem is therefore formulated as follows, where the subscript i refers to the i-th member.

$$\begin{aligned} \text{minimize} \quad & O(\mathbf{a}) = \sum_{i=1}^{m} l_i \rho_i a_i \\ \text{subject to} \quad & \sigma_i^{min} \leq \sigma_i(a_i) \leq \sigma_i^{max}, \\ & \sigma_i(a_i) \geq \sigma_i^{cr}(a_i), \\ & a_i > 0, \\ & m_{min} \leq m \leq m_{max}, \end{aligned} \tag{6.2}$$

where l_i is the length, ρ_i is the density, and a_i is the cross-sectional area of the i-th member. Furthermore, the stress limit in tension $\sigma_i^{max} \geq 0$ and the stress limit in compression $\sigma_i^{min} \leq 0$ need to be observed in each member i. All members have a circular cross-sectional areas, hence, the Euler buckling stress can be calculated as

$$\sigma_i^{cr}(a_i) = -a_i \pi E_i / (2l_i)^2 \tag{6.3}$$

and the margin of safety against buckling can be defined as

$$ms = \frac{\sigma_i^{cr}(a_i)}{\sigma_i(a_i)} - 1, \tag{6.4}$$

where $ms < 0$ indicates buckling of the respective member. Due to the academical nature of this optimization problem, the effect of initial imperfections is neglected.

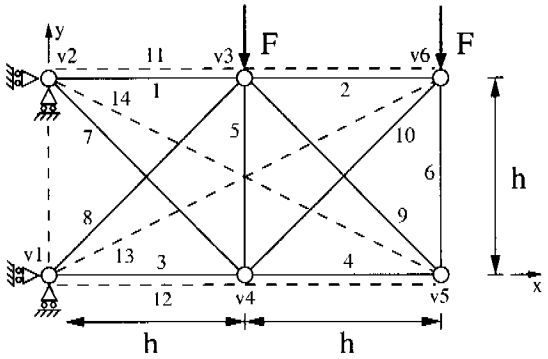


Figure 6.4: The ground structure of the ten-bar truss problem (solid lines) with five additional members indicated by dashed lines.

An EA-based optimization requires the definition of a fitness function or value, respectively, assessing the quality of each individual. For that purpose the fitness formulations presented in Section 2.5 are employed.

6.4.1.1 Results

The optimization results are compared to the results of Guo et al. [125] and Stolpe and Svanberg [140]. Thus, the same data are used for material and geometrical parameters, i.e., the height of the structure is $h = 360$, the external load is given by $F = 100$, and for the material data and the stress limits it holds $\sigma_i^{min} = -20$, $\sigma_i^{max} = 20$, $E_i = 10^4$, and $\rho_i = 0.1$ for all i.

The optimization is randomly initialized, i.e. no prior knowledge is introduced, and run with a population size of 50 individuals over 1600 generations what is a moderate number of evaluations for genetic search and such a simple model. The fast computation, processing some hundred individuals per second, leads to short optimization times. The crossover operators are applied with a general probability of 0.7, whereas the general mutation rate is set to 0.25. The probabilities of application of the single mutation and crossover operators are relatively weighted, but kept constant during the entire optimization process.

The final result of the optimization using the graph genotype concept is convincing. In Table 6.2 the result ($\tilde{\mathbf{a}}$) is compared to the published results of Guo et al. [125] ($\bar{\mathbf{a}}$) and Stolpe and Svanberg [140] ($\hat{\mathbf{a}}$). It clearly outperforms the solution of Guo et al. since its weight is significantly lower and all optimization constraints are met. Moreover, the optimization converges to the same at least local optimum as in Stolpe and Svanberg [140], whereas they solve the problem with the sequential quadratic programming package

Member	$\bar{\mathbf{a}}$	$\hat{\mathbf{a}}$	$\tilde{\mathbf{a}}$	$\check{\mathbf{a}}$
a_1	5.00000	5.00000	5.00139	12.8538
a_2	0	5.00000	5.00139	9.48097
a_3	70.35876	70.35876	70.3595	72.6613
a_4	40.62165	0	-	27.0939
a_5	57.44769	40.62165	40.6388	26.4674
a_6	40.62165	0	-	49.7662
a_7	14.14214	14.14214	14.14214	2.53085
a_8	0	0	-	-
a_9	7.07107	0	-	2.13747
a_{10}	0	68.31720	68.3187	-
$a_{11} - a_{14}$	-	-	-	-
Weight	8785.79	8553.44	8557.45	5898.23

Table 6.2: Resulting area vectors: $\bar{\mathbf{a}}$ from Guo et al. [125], $\hat{\mathbf{a}}$ from Stolpe and Svanberg [140], $\tilde{\mathbf{a}}$ from the graph genotype optimization, and $\check{\mathbf{a}}$ from the optimization with movable nodes.

SNOPT [145]. The difference in weight is approximately four grams only, hence, the optimization result marginally misses the best solution known today. A further optimization is almost impossible using EAs because the constraint values cannot exactly be reached, i.e. the tension members show tension values of $19.90 \leq \sigma_i(a_i) < 20.00$ and the compression members closely reach the Euler buckling value ($0 < ms \leq 8.4455 \cdot 10^{-4}$). Thus, the solution is almost fully stressed.

It is absolutely obvious that EAs are ill-suited for the fine tuning at the end of the optimization process since the topology can hardly be changed and only geometric and sizing parameters are adjusted. Additionally, it would be useful to alter the strategy parameters as the optimization runs, because some variation operators are more useful at the beginning when the topology is determined and some others should be forced in the end for the fine tuning. It is a matter of fact that adaptive (with regard to optimization process variables) as well as hybrid (combination of different optimization methods like stochastic search and MP) strategies could be helpful here. However, the graph genotype concept proves to be working.

6.4.1.2 Optimization including movable nodes

This example completely excludes geometry optimization since all the node positions are fixed. A further optimization is performed enabling the nodes without supports or loading, namely vertices $v4$ and $v5$, to move. The result is depicted in Figure 6.5 and the cross-sectional areas $\check{\mathbf{a}}$ of the members are also presented in Table 6.2.

The most obvious difference between this and the first solution without

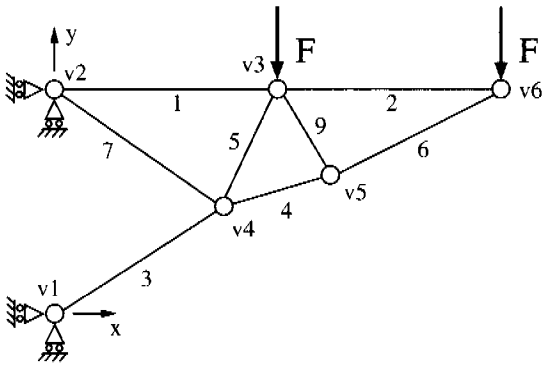


Figure 6.5: Optimization result with movable nodes. Final coordinates are $v4$: (299.62/172.99) and $v5$: (453.59/219.50)

moving nodes is the topology. One may expect that the topology would remain the same and only the geometry would be changed, but although lots of optimization runs are performed, it is impossible to find a lighter result having the same topology as the original result with static nodes. Obviously, the concurrent optimization of topology, geometry, and sizing leads to a result that could hardly be reached with sequential optimization of topology, geometry and sizing. Again, the solution is almost fully stressed, i.e., the stresses in tension bars are $19.99 \leq \sigma_i(a_i) < 20.00$ and the compression members are close to the Euler buckling value ($0 < ms \leq 2.7633 \cdot 10^{-4}$).

6.4.2 Extended planar truss optimization

The purpose of this example is to increase the complexity of the optimization problem by introducing more loaded nodes. In particular, there is no need to assume a ground structure and, thus, the variety of different topologies is not limited in advance. The problem formulation is exactly the same as shown in Equations 6.2 except for the maximum number of vertices n and the number of edges m. For this problem it holds $n = 14$, whereas the vertices $v1, v2, \ldots, v7$ cannot move, and $10 \leq m \leq 25$ what is considered to be sufficient to allow for all reasonable topologies. The external load $F = 100$ is applied five times and the loaded nodes are equidistantly distributed over the length of the structure ($h = 360$). The optimization is randomly initialized independent from any kind of ground structure and all seven movable vertices could be placed anywhere in the design space indicated by dashed lines in Figure 6.6.

6.4.2.1 Results

The optimization reveals that the number of potential vertices is chosen too high, since only three vertices are incorporated into the optimum structure also depicted in Figure 6.6. The remaining four movable vertices are isolated and therefore not depicted. In Table 6.3 a summary of the optimization result is given. For the tension members the axial stresses are listed, for the compression members the margins of safety according to Equation 6.4 are stated.

Member	$\mathbf{a}$	$\sigma_i(a_i)$	ms
a_1	146.131	-	$4.7825 \cdot 10^{-2}$
a_2	153.063	-	$4.7832 \cdot 10^{-2}$
a_3	60.0126	19.9958	-
a_4	24.1227	-	$4.7978 \cdot 10^{-2}$
a_5	62.7303	19.9975	-
a_6	73.5705	-	$4.7876 \cdot 10^{-2}$
a_7	74.5349	-	$4.7884 \cdot 10^{-2}$
a_8	16.4156	19.9974	-
a_9	33.3882	19.9970	-
a_{10}	58.7885	-	$4.8156 \cdot 10^{-2}$
a_{11}	77.4532	-	$4.7868 \cdot 10^{-2}$
a_{12}	8.17042	19.9961	-
a_{13}	18.3648	19.9949	-
a_{14}	33.1583	-	$4.8027 \cdot 10^{-2}$
a_{15}	95.5592	-	$4.7816 \cdot 10^{-2}$
a_{16}	13.3872	19.9947	-

Table 6.3: Results of the extended planar optimization example.

The structure consists of sixteen members and has an overall weight

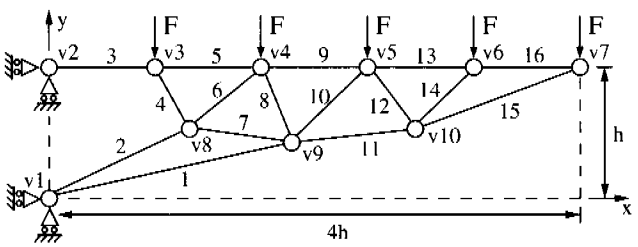


Figure 6.6: Optimum result for the extended planar truss optimization. Final coordinates are $v8$: (381.58/188.12), $v9$: (673.41/144.95), and $v10$: (981.52/188.72)

of 36704 grams. The optimization process is converged to a solution with almost fully stressed members. The axial stresses in the tension members are close to the limit and also the compression members have only a minor Euler buckling reserve of approximately 5%. The similarity of these values is caused by the fitness function definition. The compression members could definitely be pushed closer to the limit, but this would require a restart of the optimization with adjusted fitness function definition and decreased standard deviations for Gaussian mutation. Instead of using EAs for this purpose, a MP method should be used for the final fine tuning of the structure. Finally, it has to be noted that this optimization result would not have been possible with a ground structure approach only connecting neighboring nodes as in the traditional ten-bar truss problem. The triangle structure $v1, v8, v9$ could not have been generated. This result clearly shows that the graph genotype concept proves to be working for larger problems and may lead to innovative topologies.

6.4.3 Planar 52-bar truss optimization problem

6.4.3.1 Benchmark optimization

The planar 52-bar truss problem shown in Figure 6.7 is investigated to demonstrate the applicability of the graph representation to more complex optimization tasks and the results are compared to the results presented in other publications. The total mass of the structure should be minimized for the given load case, i.e., the nodes $v1$ to $v4$ are simply supported and each of the nodes from $v17$ to $v20$ is subjected to loads $F_x = 100$kN and $F_y = 200$kN. The maximum allowable stresses in tension as well as in compression members are restricted to 180MPa. Furthermore, the member areas are linked into 12 groups according to Table 6.4 and the values of the cross-sectional areas have to be chosen from Table 6.5. The Young's modulus and the density are equal for all members and their values are 2.07 x 10^5 MPa and 7860kg/m^3, respectively.

Results All the presented solutions, see Table 6.6, satisfy the constraints, hence, the total mass can directly be compared as a solution quality measure. Benchmark 1 from Wu and Chow [142] produces the worst result after 60'000 evaluations and is found by using a steady-state GA applying two-point crossovers. The results of benchmark 2 from Lemonge [143] (20'000 evaluations) and benchmark 3 from Lemonge and Barbosa [144] (population size 70, 250 generations in 20 independent runs) are obtained by applying a generational GA, whereas benchmark 3 additionally includes an adaptive penalty scheme. According to Lemonge and Barbosa [144] the resulting masses are essentially equal (1903.366392kg versus 1903.366416kg) with distinct values for some design variables. In fact, the masses are exactly equal

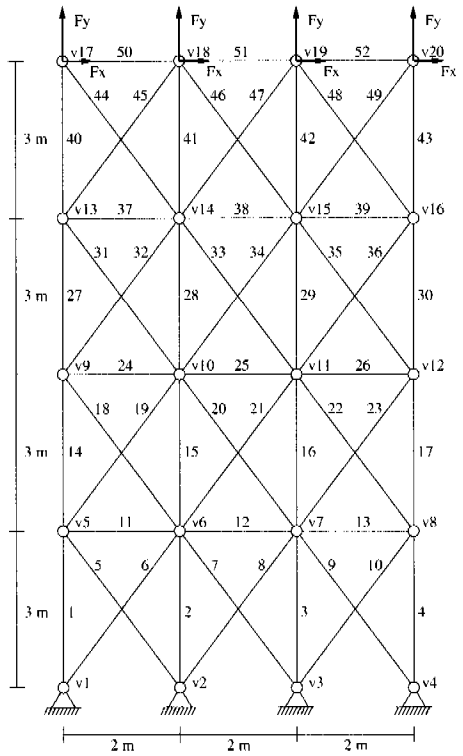


Figure 6.7: The planar 52-bar truss ground structure.

since only the order of the design variables for the groups A_3, A_6, A_9, and A_{12} is exchanged.

The graph-based truss optimization leads to a set of six similar design solutions, see Table 6.6, all of them having exactly the same mass. The first two solutions are already known from benchmark 2 and benchmark 3, but four further solutions, namely variation 1 to variation 4, having the same mass are found respecting the stress constraint. As already mentioned, the only difference of these solutions is the order of the cross-sectional areas of the horizontal members. These results are obtained by running 20 independent and randomly initialized runs with a population size of 100 graph individuals over 5'000 generations. This inefficiency can probably be explained by the fact that only sizing mutation operators and sizing crossover operators are applied since the topology and the geometry of the structure must remain unchanged. The graph representation is definitely not suited to only perform sizing optimization; the binary representation of the benchmarks 1 to 3 obviously outperforms the presented concept when applied to

Group	Members
A_1	1, 2, 3, 4
A_2	5, 6, 7, 8, 9, 10
A_3	11, 12, 13
A_4	14, 15, 16, 17
A_5	18, 19, 20, 21, 22, 23
A_6	24, 25, 26
A_7	27, 28, 29, 30
A_8	31, 32, 33, 34, 35, 36
A_9	37, 38, 39
A_{10}	40, 41, 42, 43
A_{11}	44, 45, 46, 47, 48, 49
A_{12}	50, 51, 52

Table 6.4: Member grouping for the 52-bar truss optimization problem.

No.	mm^2	No.	mm^2	No.	mm^2	No.	mm^2
1	71.613	17	1008.385	33	2477.414	49	7419.340
2	90.968	18	1045.159	34	2496.769	50	8709.660
3	126.451	19	1161.288	35	2503.221	51	8967.724
4	161.290	20	1283.868	36	2696.769	52	9161.272
5	198.064	21	1374.191	37	2722.575	53	9999.980
6	252.258	22	1535.481	38	2896.768	54	10322.560
7	285.161	23	1690.319	39	2961.284	55	10903.204
8	363.225	24	1696.771	40	3096.768	56	12129.008
9	388.386	25	1858.061	41	3206.445	57	12838.684
10	494.193	26	1890.319	42	3303.219	58	14193.520
11	506.451	27	1993.544	43	3703.218	59	14774.164
12	641.289	28	2019.351	44	4658.055	60	15806.420
13	645.160	29	2180.641	45	5141.925	61	17096.740
14	792.256	30	2238.705	46	5503.215	62	18064.480
15	816.773	31	2290.318	47	5999.998	63	19354.800
16	940.000	32	2341.191	48	6999.986	64	21612.860

Table 6.5: The available cross-sectional areas.

such optimization tasks in terms of efficiency.

Nevertheless, the graph representation is able to cope with this optimization task and finds a set of six solutions. They are not exactly identical since the maximum stresses in their members are different. All solutions are close to the upper limit regarding the tension members, but there are slight differences considering the compression members. For variation 2 the absolute value of the maximum compression stress is lower than for all other solutions and therefore this solution has to be considered as being superior to the others.

Group	Benchmark 1	Benchmark 2	Benchmark 3	Variation 1
A_1	4658.055	4658.055	4658.055	4658.055
A_2	1161.288	1161.288	1161.288	1161.288
A_3	645.160	*363.225*	*494.193*	*494.193*
A_4	3303.219	3303.219	3303.219	3303.219
A_5	1045.159	940.000	940.000	940.000
A_6	494.193	*641.289*	*641.289*	*494.193*
A_7	2477.414	2238.705	2238.705	2238.705
A_8	1045.159	1008.385	1008.385	1008.385
A_9	285.161	*494.193*	*363.225*	*363.225*
A_{10}	1696.771	1283.868	1283.868	1283.868
A_{11}	1045.159	1161.288	1161.288	1161.288
A_{12}	641.289	*494.193*	*494.193*	*641.289*
W	1970.142	1903.3664	1903.3664	1903.3664
σ_{tmax}	178.924	179.906	179.897	179.783
σ_{cmin}	-153.109	-165.81	-165.143	-155.272
Group	Variation 2	Variation 3	Variation 4	Buckling
A_1	4658.055	4658.055	4658.055	5999.998
A_2	1161.288	1161.288	1161.288	3703.218
A_3	*494.193*	*494.193*	*363.225*	1993.544
A_4	3303.219	3303.219	3303.219	3703.218
A_5	940.000	940.000	940.000	3703.218
A_6	*363.225*	*363.225*	*494.193*	2180.641
A_7	2238.705	2238.705	2238.705	2180.641
A_8	1008.385	1008.385	1008.385	3096.768
A_9	*494.193*	*641.289*	*641.289*	1993.544
A_{10}	1283.868	1283.868	1283.868	792.256
A_{11}	1161.288	1161.288	1161.288	2477.414
A_{12}	*641.289*	*494.193*	*494.193*	1890.319
W	1903.3664	1903.3664	1903.3664	3782.8242
σ_{tmax}	179.962	179.987	179.963	169.468
σ_{cmin}	-150.432	-165.484	-165.848	-79.296

Table 6.6: Comparison of the 52-bar planar truss optimization results. Benchmark 1: Wu and Chow [142], benchmark 2: Lemonge [143], and benchmark 3: Lemonge and Barbosa [144]. The cross-sectional areas of the horizontal members are printed in italic shape for the six variations of equal mass. For each solution the total mass, the maximum tension stresses as well as the maximum compression stresses are stated.

Consideration of Euler buckling From an engineering point of view it is absolutely mandatory to perform a buckling analysis since the truss members are slender. For this analysis the Euler buckling criterion according to Equation 6.4 is applied, which reveals that none of the previously presented truss structures is stable. Consequently, the following truss optimizations

are extended by an Euler buckling constraint analogously to the examples of Sections 6.4.1 and 6.4.2. The additional Euler buckling constraint in combination with the discrete design variables complicates the optimization task. A lot of individuals violating the buckling constraint are created by the operators since choosing the next smaller cross-sectional area from Table 6.5 often leads to buckling configurations. Therefore, 20 independent optimizations with population size 100 are run over 10'000 generations leading to the optimum design with a mass of 3782.8242kg, see Table 6.6. The solution fulfilling the Euler stability criteria is almost 100% heavier than the results of the previously presented optimizations.

6.4.3.2 Free optimization

The planar 52-bar truss optimization problem is reformulated in order to demonstrate the full capacity of the graph representation. So far, the topology as well as the geometry of the 52-bar truss could not be changed and the design space was further reduced by the prescribed grouping of the member cross-sectional areas. Actually, these restrictions inhibit the optimizer from discovering innovative design solutions. Thus, the optimization task is formulated in a more flexible manner, i.e., only the supported and the loaded nodes of the truss are given. All other nodes are not restricted to any location and can move through the design domain, i.e. the rectangle defined by the vertices $v1$, $v4$, $v12$, and $v9$ in Figure 6.8. The maximum number of nodes is chosen to be 20 and the number of members must be in between 10 and 60. The optimization formulation is analogous to the definition in Equations 6.2 so that the maximum stress constraint as well as the buckling constraint have to be fulfilled.

The complexity of this optimization task is considerably higher than before and requires therefore many more evaluations in order to converge to a solution. First, the population size is set to 500 and the optimization is run over 10'000 generations. Afterwards, the best solution of the first run is taken as initial design for a second optimization run (population size 250 over 4'000 generations) with adjusted optimization parameters in order to emphasize the fine tuning of the structure. After six million evaluations the solution depicted in Figure 6.8 is found with a final mass of 1097.7158kg. The cross-sectional areas, the maximum tension stress values, and the margins of safety for buckling of each member are within the limits and can be found in Table 6.7. The resulting design consists of 16 members whereof nine are tension members and seven are under compression and none of the compression members reaches critical stresses close to the limit of 180 MPa.

Only two of the four supported nodes are used and the members are quite logically arranged. In general, the compression members are kept rather short in order to fulfill the buckling constraint with relatively slender members. Furthermore, the sequence of the compression members 5

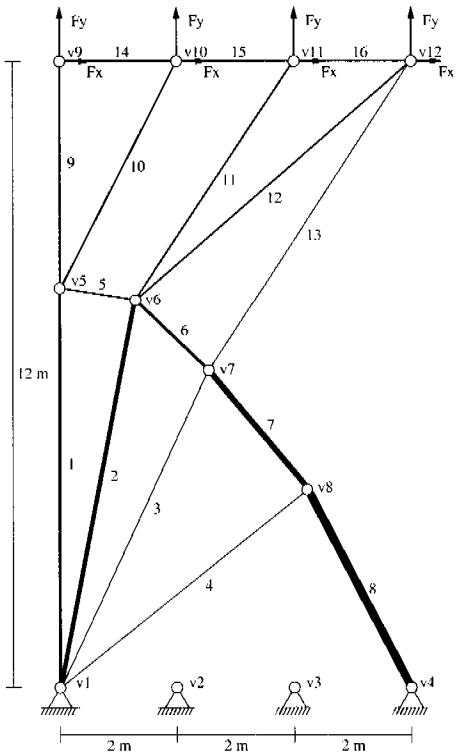


Figure 6.8: Optimum result for the planar truss optimization. Final coordinates in mm are $v5$: (0.0/7665.7), $v6$: (1306.4/7365.3), and $v7$: (2502.4/6088.6), $v8$: (4204.0/3799.1)

to 8 approximates a parabolic shape what seems to be optimal to block the rotation of the structure around the node $v1$. The resulting design solution is approximately 71% lighter compared to the design solution with buckling constraint presented in Table 6.6. This convincing optimization result clearly shows that the graph representation of truss structures is able to fit sophisticated structures into given boundary conditions, although the computation costs are still rather high.

6.5 Conclusion

In the scope of this chapter a novel graph-based representation concept is introduced allowing for the optimization of truss structures with EAs. The concurrent optimization of topology, geometry, and sizing is a central property of the presented representation approach. The optimization examples

demonstrate the quality of the method leading to innovative topologies independent from any kind of ground structure. Basically, only the loaded and clamped nodes must be given and the optimization method itself tries to fit an optimum topology with optimum geometry and sizing into the given design space.

Nevertheless, the concept needs to be further developed in terms of adaptivity and fine tuning. Practice has shown that the fine tuning towards the end of the optimization is expensive. In order to speed up this process the standard deviation parameters of Gaussian mutation are manually reduced. Furthermore, the operator rates of topology operators which hardly can produce improved solutions when the optimization is almost converged to an optimum are also manually reduced. The optimization algorithm should therefore be extended with self-adaptive mechanisms adjusting the Gaussian mutation parameters as well as adapting the operator rates of different operator types according to the current search state. A possible approach to adaptive variation operator rates is presented in Chapter 4, but this concept would need an adaptation to graph genotypes.

Finally, EAs are perfectly suited to sample a multimodal search space, but their fine tuning capabilities are limited compared to gradient-based optimization algorithms. Consequently, a further extension could be a hybrid optimization algorithm combining the traits of genetic search methods and MP in order to accelerate the optimization process providing the same or even better result quality.

Member	**a**	$\sigma_i(a_i)$	ms (Equation 6.4)
a_1	2341.191	179.91837	-
a_2	3703.218	175.91492	-
a_3	494.193	172.49449	-
a_4	494.193	176.52827	-
a_5	1045.159	-	$4.36602 \cdot 10^{-2}$
a_6	2896.768	-	0.12584
a_7	4658.055	-	$1.88401 \cdot 10^{-2}$
a_8	6999.986	-	$1.96118 \cdot 10^{-2}$
a_9	1161.288	172.22227	-
a_{10}	1283.868	171.56366	-
a_{11}	1374.191	168.33515	-
a_{12}	1374.191	179.91868	-
a_{13}	198.064	154.17152	-
a_{14}	1690.319	-	0.16130
a_{15}	1690.319	-	$7.80826 \cdot 10^{-2}$
a_{16}	1535.481	-	$4.75843 \cdot 10^{-2}$

Table 6.7: Results of the free optimization. The final mass is 1097.7158kg.

Chapter 7

Graph-based global laminate optimization

This chapter is based on the paper draft submitted to Structural and Multidisciplinary Optimization: *A graph-based parameterization concept for global laminate optimization* [146].

7.1 Introduction

Today composite laminates are popular in aerospace, marine, automotive, and sport applications due to their excellent mechanical properties as well as the adjustability of these through the combination of different laminae. However, an optimal layout of a composite structure is still hard to find: complex geometries, different materials, and manufacturing as well as economical limitations typically lead to highly constrained problems with many parameters.

A lot of research has been done in the field of composite optimization. A thorough review can be found in Venkataraman and Haftka [67]. The representation is one of the most important issues since it strongly influences the quality of the optimized structure. There are a few different approaches for the parameterization of laminate optimization problems. The most simple one is the representation of the stacking sequence by a material and an orientation parameter for each layer [147, 148]. A further refinement considers the number of layers as an optimization parameter what leads to a variable number of parameters for different design solutions. Apparently, this representation suffers from its global point of view because a single stacking sequence holds for the entire laminate, i.e. the entire mechanical structure.

Most often, structural failure and constraint violations are local effects, hence, an optimal laminate representation should allow for local reinforcements. A representation which divides the design space into different regions (cells) with individual laminate properties overcomes this disadvantage. The

laminates of adjacent cells should have some common layers to ensure the cohesion of the entire structure. Thus, the problem is to encode the stacking sequence and the shape of these regions without introducing an unnecessary large number of additional constraints.

The *patch concept* proposed by Zehnder and Ermanni [121] which is already introduced in Section 5.1.2 convinces with its manufacturing-oriented approach whereas the shapes of the patches are modeled using CAD software. In Chapter 5 the patches are generally of rectangular shape and are mapped to an underlying FE-mesh in the mapping stage. Two drawbacks of this method can be identified. First, the patches are limited to a predefined, e.g. rectangular, shape which is inappropriate in certain situations. The second issue is the mapping stage which is not perfectly injective. Patches crossing the design space boundaries and patches with slightly different parameter values may lead to identical phenotype solutions due to the FE-discretization. In order to overcome these drawbacks a completely new developed graph-based patch and laminate representation is introduced representing the FE-mesh with a mathematical graph abstraction. A single patch is represented by a subgraph including only a subset of all finite elements, whereas such a subgraph is connected to an additional virtual vertex holding the properties, e.g. material and orientation, of the respective patch. This approach allows for almost arbitrary patch geometries and substantially improves the injectivity property of the mapping process.

The graph-based representation concept is explained in Section 7.2. Section 7.3 gives an overview of the evolutionary operators particularly adapted to the presented genotype and Section 7.4 shortly discusses some implementation details. Finally, two engineering problems illustrate the performance and capabilities of the new method in Section 7.5 before the chapter is concluded with Section 7.6.

7.2 Graph-based laminate representation

7.2.1 Graph abstraction of FE-meshes

A commonly used method for the numerical simulation of the mechanical behavior of composite laminates is FEM [149]. The FE-mesh consists of elements and their associated nodes, whereas its structure, e.g. the number of nodes per element or the shape of the elements, depends on the element type(s) used. In this chapter only the special case of composite laminates is considered. Such composite structures are usually thin-walled, three-dimensional, and laminar shapes which can be modeled with shell elements. Special types of these shell elements are able to model a layered laminate assembly and are commonly used in a triangular and a rectangular implementation with six or eight nodes, respectively.

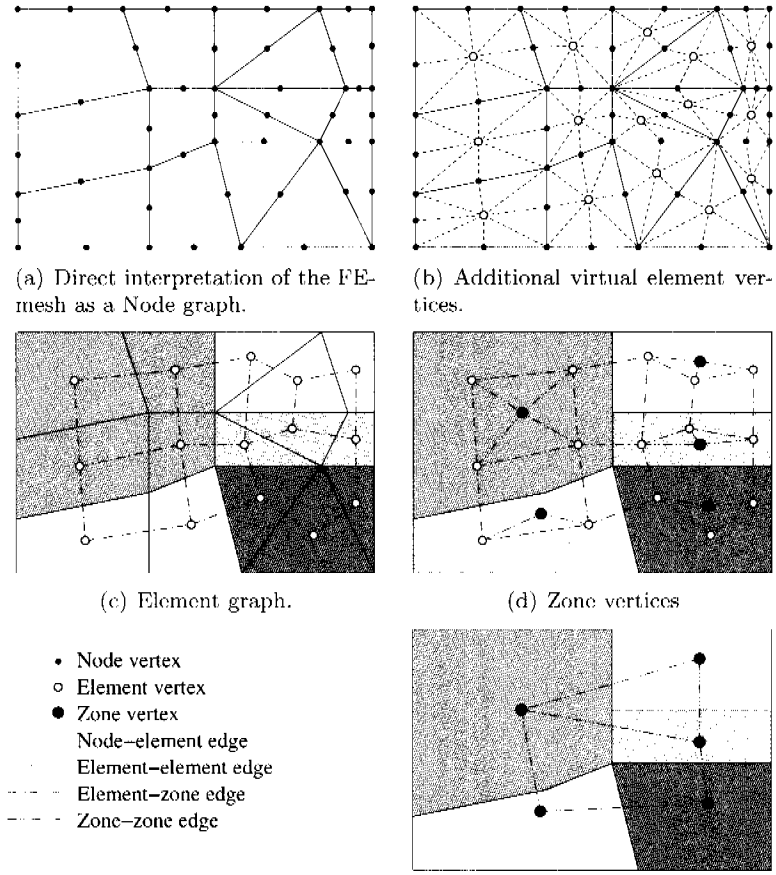


Figure 7.1: Build-up of the zone graph.

The information included in such a FE-mesh can be transfered into different graph abstractions as it is illustrated on a basic two-dimensional FE-mesh consisting of fifteen elements in Figure 7.1. The *node graph* in Figure 7.1(a) represents the adjacency information of the mesh nodes, whereas a vertex represents a FE-mesh node and neighboring nodes are connected by an edge. A second graph shown in Figure 7.1(b) represents the finite elements as additional vertices and stores contiguity information between elements and their nodes. Then, the *element graph* illustrated in Figure 7.1(c) can be built which contains the element adjacency information of the FE-mesh. The granularity of the FE-mesh is most often chosen rather small in order to achieve a sufficient simulation accuracy. Consequently, an additional abstraction layer is introduced which groups several adjacent finite elements to so-called *zones*. In Figure 7.1(c) the differently shaded areas in-

dicate a possible grouping of finite elements to a total of five zones. For each zone a virtual *zone vertex* is introduced which is connected to the element vertices belonging to the respective zone as illustrated in Figure 7.1(d). Finally, the zone vertices of adjacent zones are connected by zone-zone edges, see Figure 7.1(e), in order to store neighborhood information on the zone level. The laminate optimization method presented in this chapter directly operates on this zone graph. Subsequently, several definitions are given to clarify the concept of the element and the zone graphs.

Def. 1 (Element vertex) *An element vertex is an abstraction of one finite element. It points at the underlying finite element and stores information on the element's nodes.*

Def. 2 (Element graph) *The element graph contains the element vertices of all finite elements. Two element vertices of finite elements with more than one coincident mesh node are connected with an element-element edge.*

Def. 3 (Zone vertex) *A zone vertex is an abstraction of a group of connected element vertices. It points at the element vertices of the respective elements by element-zone edges, whereas an element vertex can only belong to a single zone.*

Def. 4 (Zone graph) *The zone graph consists of all zone vertices. Two neighboring zones pointing at adjacent element vertices are connected with a zone-zone edge.*

The definition of the zones is completely user-defined and should be adapted to the optimization problem. A zone can group thousands of elements, but it is also possible that a single zone is assigned to each finite element.

7.2.2 The graph-based laminate representation

First, the representation of a single patch is presented before the representation of the entire laminate is addressed.

7.2.2.1 Patch representation

According to Zehnder and Ermanni [121] a patch is defined as follows:

Def. 5 (Patch) *A patch is one global layer of a laminated structure.*

The following requirements should be fulfilled by the genotype representation of a patch:

- Discretization: In the phenotype space one finite element is the smallest inseparable building block of a patch. This is caused by the discretization with layered elements. Once the FE-mesh is defined, a genotype offering higher resolution than this FE-mesh affects the optimization efficiency. A flexible mesh would demand remeshing of the structure before every evaluation what increases the computational costs for each evaluation. Therefore, this graph-based laminate representation follows a fixed-mesh approach which maps each genotype to an identical FE-mesh to produce the respective phenotype solution. Thus, a set of dedicated zones and the underlying finite elements, respectively, incorporates the shape, the size, and the position of one patch.
- Properties: Several properties which can be considered as coupled or independent optimization parameters, e.g. material, orientation, thickness, etc., can be assigned to a patch.
- Connectivity: A patch is a connected region on the shape of a mechanical structure. Its dimensions are limited by the boundary of the mechanical structure and possibly by further constraints like a maximum number of zones or finite elements belonging to the respective patch.

These requirements can be satisfied with the following approach: The genotype is based on the zone graph of the mechanical structure. This zone graph stays the same during the optimization, i.e., the design domain is unchanged and no remeshing is required during the optimization process. The zone graph is enhanced with a virtual *patch vertex*.

Def. 6 (Patch vertex) *A patch vertex is an abstract entity representing the properties of one patch which points at an arbitrary number of zone vertices. It takes a heterogeneous set of representation-dependent genes to encode the properties of the respective patch such as material, orientation, etc.*

The representation-dependent genes are typically chosen from the list of genes included in the *UniGene* concept (cf. Appendix A).

Each zone vertex within the shape of one patch is connected with an edge to the newly introduced patch vertex as shown in Figure 7.2. The connectivity of the patch shape is not guaranteed, it has to be achieved with functional refinements.

Def. 7 (Patch graph) *A subgraph including all adjacent zone vertices of a patch vertex and their connecting edges is called patch graph.*

Def. 8 (Patch chromosome) *The information represented by a patch vertex in conjunction with its affiliated genes and its adjacent edges encodes one physical patch and is further on called patch chromosome.*

During the mapping stage the properties stored in the patch chromosome are assigned to the finite elements which belong to the respective patch. In general, the laminate structure consists of a variable number of patches, whereas the order of the patches still needs to be defined.

7.2.2.2 Laminate representation

A laminated structure is represented as a set of patch chromosomes organized in a vector. The order of the patches in the variable-length genotype vector encodes the stacking sequence of the laminate, i.e., the mapping from the genotype to the phenotype solution processes the set of patches always in the same order. The number of patches can be changed during the optimization process since the genotype vector possesses variable-length properties. Such a genotype contains information on three levels:

- Laminate level: describes the set and the sequence of the patches analogous to a global lamination plan. It is represented by the variable-length genotype vector.
- Patch level: one patch chromosome encodes the size, the shape, and the position of a patch.
- Patch property level: contains the properties of one patch. They are encoded as genes in a patch vertex.

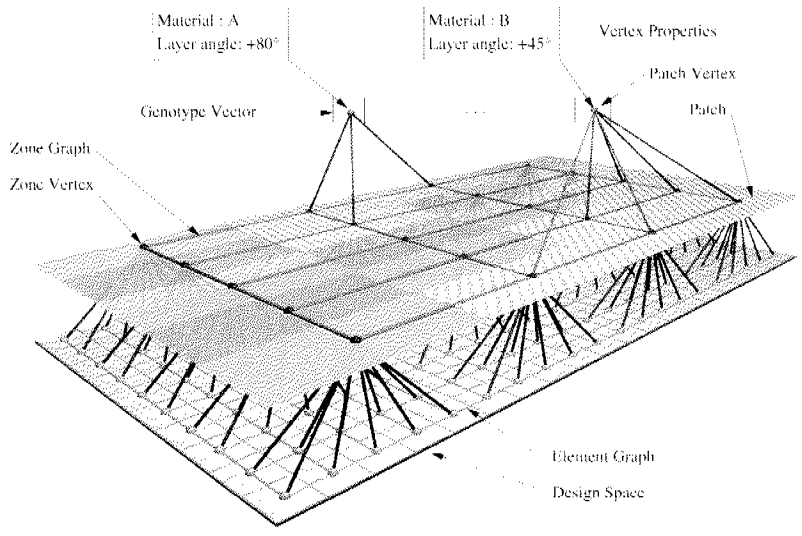


Figure 7.2: Zone representation of two patches.

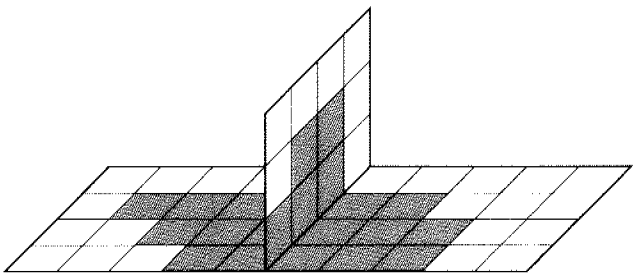


Figure 7.3: Splitted patch at three coincident surface boundaries.

In Figure 7.2 an illustrative example of the laminate representation is shown. The patch vertices are connected to the appropriate zone vertices which are themselves further connected to the element vertices belonging to the respective patch. During the mapping process all patch properties which are stored in the patch vertex are transfered to the underlying finite elements. Typically, a zone is covered by several patches what finally leads to finite elements with multiple layers.

The graph-based laminate optimization method is limited to surfaces consisting of simply connected areas like composite panels. More complex structures, e.g. beams with T-cross-sections, cannot be entirely optimized yet, because three or more surfaces have coincident boundaries. Depending on the normal direction along which the stacking sequence is evaluated, this could lead to intersecting layers. Moreover, patches crossing coincident boundaries of at least three independent surfaces could be split as shown in Figure 7.3 what is impossible from a manufacturing point of view. However, such adjacent surfaces could be optimized independently.

After defining the patch genotype some patch manipulation rules need to be defined which are necessary to create new design solutions.

7.2.3 Patch structure manipulation

All structural patch manipulations can be interpreted as either removing or adding edges between a patch vertex and some zone vertices in the zone graph. In general, each modification changes the patch graph and thus the shape of the respective patch. The graph abstraction of the patch allows for an efficient analysis of the patch graph in terms of connectivity, size, shape, and position.

7.2.3.1 Regions of a patch

Adjacency information allows to specify different regions of a patch as shown in Figure 7.4:

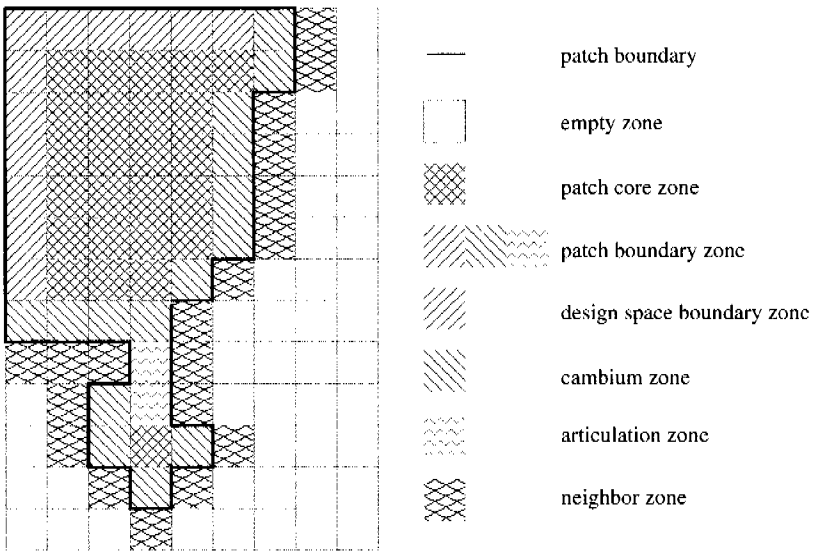


Figure 7.4: Illustration of different patch regions on a rectangular design space with rectangular zones.

- *Patch zones* are all zones identified by their zone vertices which belong to the patch graph.
- *Patch core zones* are zone vertices in the patch graph which are completely surrounded by other zones from the same patch.
- *Boundary zones* are patch zones with a reduced set of adjacent patch zones: *Design space boundary zones* have no adjacent zones outside of the patch, *cambium zones* have adjacent elements outside of the patch and *articulation zones* connect two regions of a patch graph. Removing one articulation zone splits the patch graph into different components.
- *Neighbor zones* are not member of the patch, but are adjacent to at least one patch zone.

7.2.3.2 Basic growth and shrinking mechanisms

Growth and shrinking operations are zonal patch variations. They are basic components of the variation operators presented in Section 7.3 and their application in stochastic search algorithms demands a random behavior.

Engineering as well as manufacturing require a patch shape to stay *compact* and *connected*. To ensure this, growth mechanisms are limited to

add neighbor zones and shrinking mechanisms only remove boundary zones, whereas articulation zones are handled in a special way.

The *neighborhood size* concept is introduced to describe the environment of zone vertices.

Def. 9 (Neighborhood size) *The neighborhood size $\eta(z, \pi)$ of a zone vertex z in respect of the patch π is the number of its adjacent patch zones.*

Def. 10 (Relative neighborhood size) *The relative neighborhood size $\rho(z, \pi)$ of a zone vertex z in respect of the patch π is its neighborhood size $\eta(z, \pi)$ divided by the number of possible adjacent zones p.*

Accordingly, the number of *neighbor zones* of a given zone of patch π is

$$\upsilon(z, \pi) = p - \eta(z, \pi) \tag{7.1}$$

As a selection criterion for zones to be removed or added to a patch the neighborhood size concept is the basis for compact patch shapes. Growth and shrinking mechanisms use a random weighted selection procedure, where the weight factors depend on the relative neighborhood size of the respective zones.

Parallel growth mechanism The parallel growth mechanism resizes a patch by adding a predefined number of edges between a patch's neighbor zones and its patch vertex. The method performs a random weighted selection out of all neighbor zones, where the weight for each neighbor zone is equal to its relative neighborhood size $\rho(z_i, \pi)$.

Parallel shrinking mechanism The parallel shrinking mechanism works analogously to the parallel growth mechanism. It removes edges between a patch's boundary zones and its patch vertex. There are two variations of the method: The first one randomly selects out of a pool of all boundary zones, the second one's pool contains all boundary zones except for articulation zones. The selection weight is one minus the relative neighborhood size $\rho(z_i, \pi)$ of the respective zone. The first variation does not protect the connectivity of the patch shape.

Recursive growth mechanism While the parallel mechanisms perform variations on the whole shape of a patch, the recursive methods work more locally. Neighbor zones of a ρ-proportionate selected cambium zone are added to the patch with a certain probability. In case a neighbor zone is newly connected to the patch this zone becomes a cambium zone of the patch and a new instance of the growth algorithm is started at this zone with lower probability for further growth.

A probability $p_g(z,i)$ for an arbitrary cambium patch zone z depending on the recursion depth i is defined as

$$p_g(z,i) = e^{-d_g \cdot \frac{i}{\rho(z,\pi)}}, \tag{7.2}$$

where d_g is an user-defined growth parameter. This probability defines the growth behavior of a patch which is outlined in Algorithm 3. The number of zones per patch n_z can theoretically grow until the patch reaches the boundary of the design space. However, normally this is improbable since p_g diminishes fast with increasing recursion depth i. Alternatively, a maximum number of zones per patch n_z^{max} can be given as termination criterion.

Algorithm 3 Recursive patch growth algorithm.

1: Choose a cambium zone z by ρ-proportionate selection.
2: Initialize recursion counter $i = 0$.
3: **for** $a = 0$, $a < v(z,\pi)$ **do**
4: **if** $n_z \geq n_z^{max}$ **then**
5: Terminate.
6: **end if**
7: Make a random choice with probability $p_g(z_a,i)$ for $\beta = true$ out of [*true, false*], where z_a is the a-th neighbor zone of z.
8: **if** $\beta = true$ **then**
9: Add an edge between the patch vertex and the zone z_a.
10: Start a new instance at 3 for the newly connected zone $z = z_a$ and $i = i + 1$.
11: **end if**
12: $a = a + 1$
13: **end for**

Recursive shrinking mechanism The recursive shrinking mechanism works analogously to the recursive growth method. First, a ρ-proportionate selected boundary zone is removed from the patch. All adjacent zone vertices which still belong to the patch are then additionally removed with a certain probability $p_s(z,i)$ which is defined as

$$p_s(z,i) = e^{-\frac{i}{d_s \cdot (1-\rho(z,\pi))}}, \tag{7.3}$$

where d_s is an user-defined shrinking parameter. The probability decreases rather fast with the recursion depth i so that it is improbable that a patch completely vanishes. Additionally, a minimum patch size, i.e. an user-defined minimum number of zones n_z^{min}, can be defined in order to avoid small patches. If the number of zones n_z reaches this minimum number, the shrinking mechanism is stopped.

The original shrinking mechanism is outlined in Algorithm 4 and a variation of the method ensures the connectivity of the patch graph by excluding articulation zones from the selection pool.

Algorithm 4 Recursive patch shrink algorithm.

1: Choose a boundary zone z by ρ-proportionate selection.
2: Initialize recursion counter $i = 0$.
3: Remove the edge between the patch vertex and the zone z.
4: **for** $a = 0$, $a < \eta(z, \pi)$ **do**
5: **if** $n_z \leq n_z^{min}$ **then**
6: Terminate.
7: **end if**
8: Make a random choice with probability $p_s(z, i)$ for $\beta = true$ out of [*true, false*], where z_a is the a-th adjacent patch zone of z.
9: **if** $\beta = true$ **then**
10: Start a new instance at 3 for the zone $z = z_a$ and $i = i + 1$.
11: **end if**
12: $a = a + 1$
13: **end for**

7.3 Graph variation operators

For the optimization of composite laminates with the graph-based representation mutation as well as crossover operators need to be defined. Since the genotype groups genetic information on different levels, namely laminate, patch, and patch property level, there are different types of operators addressing each of these levels.

7.3.1 Mutation Operators

7.3.1.1 Laminate mutation operators

- *Add patch mutation* inserts a randomly initialized patch chromosome at a random position in the vector genotype.
- *Remove patch mutation* removes a randomly chosen patch chromosome from the vector genotype.
- *Swap patch mutation* swaps two patch chromosomes in the genotype. This changes the stacking sequence of the laminate.

7.3.1.2 Patch mutation operators

- *Parallel resize mutation* resizes a patch to a randomly chosen size by either applying parallel growth or parallel shrinking.

- *Recursive resize mutation* resizes a patch by recursive growth and shrinking mechanisms, whereas the final number of zones within the patch results from the random termination of the resizing mechanism.
- *Random walk mutation* removes a random boundary zone z_b from a patch and adds a random neighbor zone z_n which is close to z_b. The zone z_b is randomly chosen from all boundary zones, z_n is the exit point of a random walk through the patch graph.
- *Bobtail mutation* renders a patch shape more compact by removing one randomly chosen articulation zone and the smaller of the remaining components from a patch.

7.3.1.3 Property mutation operators

Uniform and Gaussian mutation operate on patch vertex properties. Typically, these patch vertex properties are encoded with universal genes (cf. Appendix A) providing these standard mutation operators.

7.3.2 Crossover Operators

7.3.2.1 Laminate crossover operators

- *Uniform crossover* takes two parent individuals and swaps two randomly chosen chromosomes a predefined number of times.
- *One point crossover* takes two parent individuals and selects a random crossover point for each. The head of the first individual is recombined with the tail of the second one and vice versa.
- *Two point crossover* takes two parent individuals and selects two random crossover points for each. The segments of both individuals between the crossover points are exchanged.
- *Intermediate crossover* takes two parent individuals and randomly exchanges patch chromosomes with a given probability, but keeps the stacking sequence as shown in Figure 7.5.

7.3.2.2 Patch crossover operators

- *Marry crossover* takes two parent individuals and creates one offspring which is based on the first parent. It chooses two patch chromosomes out of its parents at random and searches for the shortest path between the two patch vertices of the chromosomes. Each zone of the second parent's chromosome and on the connecting path are added to the first parent. In a second step a recursive growth is started on each

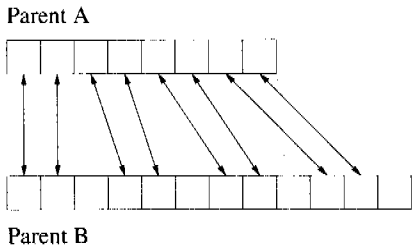


Figure 7.5: Possible chromosome exchanges in intermediate crossover.

zone on the path to create a more compact shape, whereas the growth rate is user-defined. The modified first parent becomes the offspring individual.

- *Position crossover* takes two parent individuals and creates one offspring which is based on the first parent. It searches again for the shortest path from a randomly chosen chromosome of the first parent to a chromosome of the second parent. From the middle of this path a new patch grows (recursive and parallel growth) to a random size in between the size of the two parent chromosomes. The respective patch of the first parent is replaced by the newly created patch and the modified first parent becomes the offspring individual.

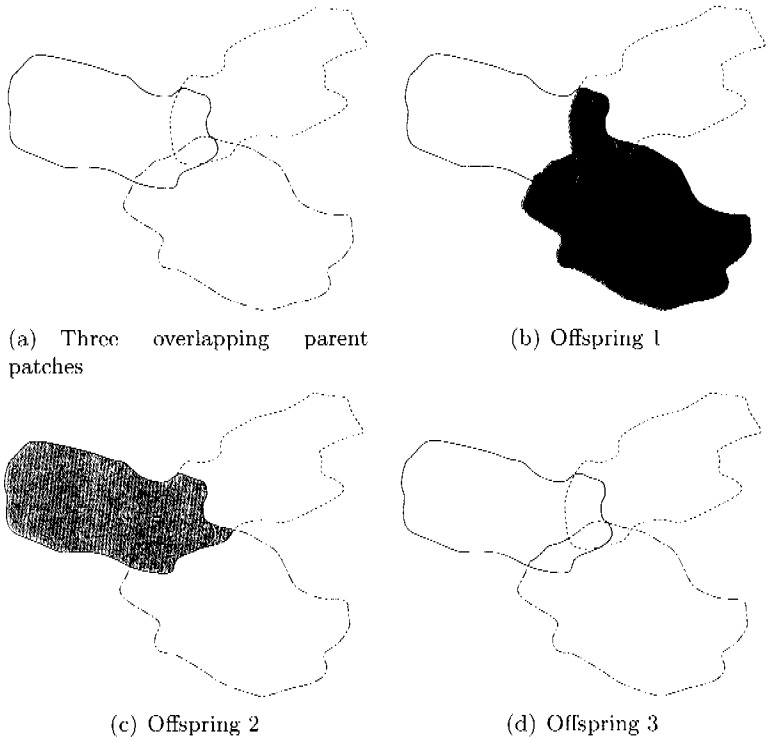
- *Overlap crossover* selects three parent individuals and creates three offspring as illustrated in Figure 7.6. The parent selection searches for three parent individuals having overlapping patches (A, B, C), whereas patch A belongs to the first, patch B to the second, and patch C to the third parent, respectively. The overlapping parts of patch B and C are added to patch A, those of A and C to patch B, and those of A and B to patch C. Afterwards, the modified parents become the offspring. This way the operator exchanges geometric subsets between patches even on curved shapes.

7.3.2.3 Property crossover operators

Segment crossover is an arithmetical crossover determining new gene values according to Equation 2.7 for randomly chosen patch vertex properties.

7.4 Implementation

The basic EA-functionalities are taken from the Evolving Objects library [93] (EOlib) and the graph data structures and graph algorithms derive from the



(a) Three overlapping parent patches

(b) Offspring 1

(c) Offspring 2

(d) Offspring 3

Figure 7.6: Overlap crossover. Resulting patches appear shaded.

Boost Graph Library [139] (BGL). The patch property genes are a part from the *eoUniGene* library, see Appendix A, and the FE-analysis is performed with FELyX [150]. Therefore, the mapping process from the graph-based laminate representation to the FE-model is straightforward since the vertices and the finite elements are directly accessible. Furthermore, all involved libraries and further program code are written in C++ what leads to a high-performance optimization environment.

7.5 Application

7.5.1 Eigenfrequency optimization of a rectangular plate

Some simple tests with the eigenfrequency optimization of a rectangular plate known from Section 5.3 are carried out in order to demonstrate the influence of different zone layouts. The material of the patches as well as the representation of the patch orientations in discrete steps of 5° is identical.

7.5.1.1 Optimization setup

Two different zone layouts A and B are compared to each other. The FE-model shown in Figure 5.4 consists of a total of 200 finite elements. For the first layout A a single zone is assigned to each finite element, i.e., each finite element can theoretically be assigned with an individual stacking sequence. The second layout B is based on the zone configuration shown in Figure 7.7. This layout includes much more domain knowledge compared to layout A, but the local adaptability is reduced. The zones around the concentrated mass are smaller since there a local reinforcement contributes much more to the maximization of the first eigenfrequency than a reinforcement on the right side of the rectangular design domain. Although these zones still group several finite elements, it is rendered possible to locally arrange relatively small reinforcement patches. Obviously, the lower number of zones of layout B leads to a decisively smaller search space.

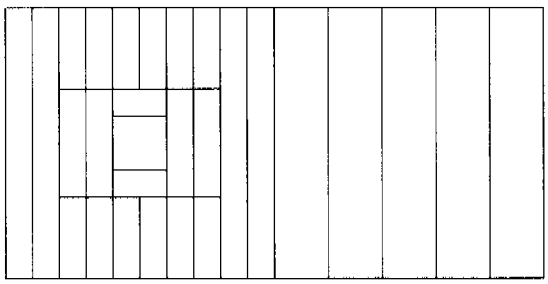


Figure 7.7: Zone layout B. The FE-mesh is hidden for better perceptibility.

The optimization objective is the maximization of the first eigenfrequency subject to a mass constraint which is set to 0.48275kg. Due to the fitness function definition also slightly heavier solutions up to 0.53275kg are possible, but they are penalized. For better comparability the number of patches is set to twelve patches for both zone layouts.

7.5.1.2 Results

Several optimization runs for both zone layouts are performed. Only the best optimization runs are presented here, since the differences in terms of optimization efficiency and result quality are enormous. Due to the fact that the search space sizes of these optimization tasks are different the population sizes are chosen unequal. A generational EA with the weak elitism mechanism is employed, whereas the basic strategy parameters, namely global mutation (0.4) and crossover probabilities (0.7) as well as the tournament selection with tournament size two, are equal.

The best of eight runs of zone layout A reaches an eigenfrequency of 27.8534Hz with a final mass of 0.50090kg. For this result shown in Figure 7.8 a total of 500'000 evaluations is needed, i.e. 10'000 generations are evaluated with a population size of 50 individuals. The patch shapes are rather frayed what is unfavorable from a manufacturing point of view. It is quite obvious that the number of zones is chosen too large for such a simple problem. The orientation of most of the patches is almost vertical what obviously contributes to an increased eigenfrequency.

The best run of zone layout B produces a solution with an eigenfrequency of 30.3468Hz and a mass of 0.50135kg. The eigenfrequency value is significantly higher after only 200'000 evaluations ($P = 100$ and 2'000 generations) than the solution of zone layout A, whereas the mass constraint is slightly more violated. Obviously, the patch shapes are much tighter and manufacturing-oriented due to the zone layout as can be seen in Figure 7.8. Furthermore, the zone covering the region of the concentrated mass is most often blank because the mass itself slightly stiffens the respective zone.

As expected, the subdivision of the design domain into zones has a major impact on the optimization efficiency and result quality. If a lot of relatively small zones are defined, the 'degree of freedom' is enormous. Accordingly, the optimization process requires a large number of evaluations in order to converge to an at least local optimum solution. It is therefore of great importance to define a reasonable zone layout which is tailored to the optimization problem.

Subsequently, a sailplane wing is optimized in order to demonstrate the applicability of the presented graph-based representation concept to laminated structures.

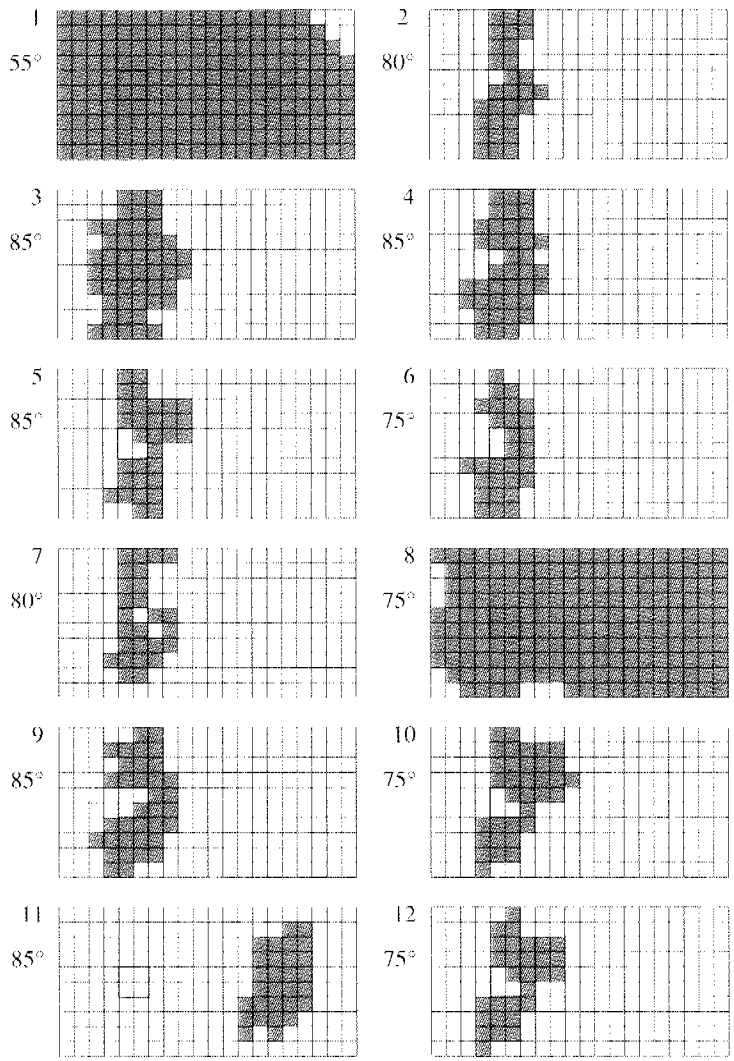


Figure 7.8: Resulting twelve patches of zone layout A.

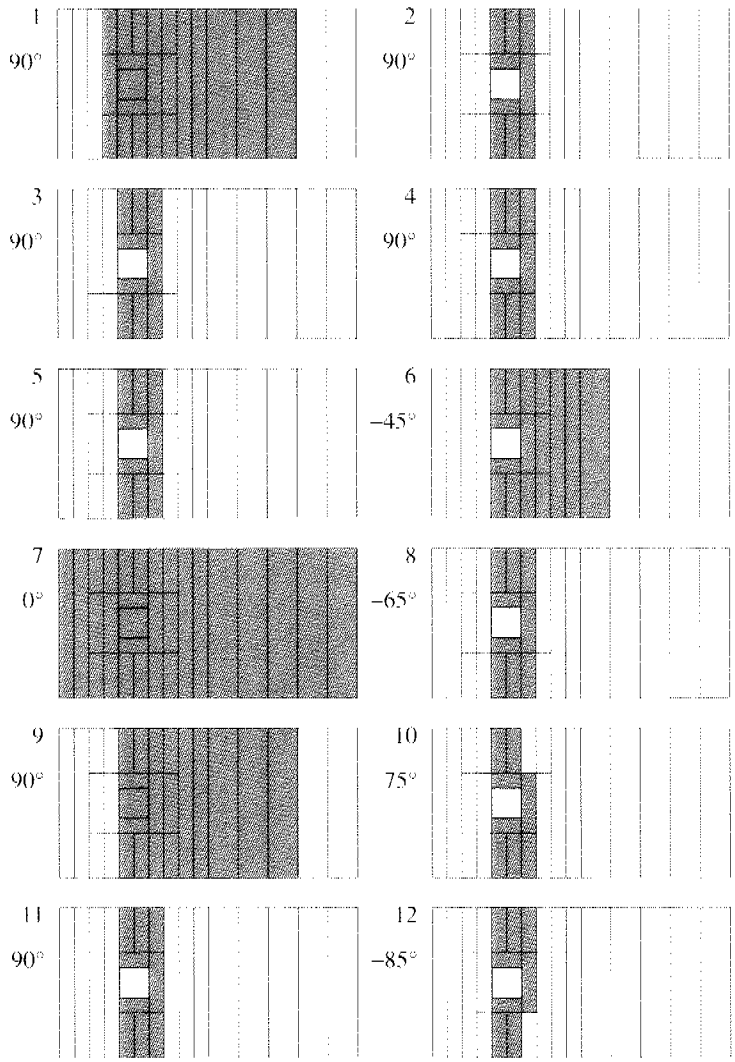


Figure 7.9: Resulting twelve patches of zone layout B.

7.5.2 Sailplane wing optimization

7.5.2.1 Problem description

The wing structure is inspired by the DG-1000, a two seated sailplane built by *DG-Flugzeugbau*[1]. The characteristic values of the sailplane are given in Table 7.1.

wingspan	20	[m]
wing area	17.51	[m^2]
aspect ratio	22.84	[-]
basic weight	410	[kg]
maximum airspeed	270	[km/h]

Table 7.1: Technical data of the DG-1000 sailplane.

The wing structure is manufactured from composite materials as illustrated in Figure 7.10. The yellow layers are fiber glass which are common in sailplane structures and the green material is a foam which is necessary for the construction of sandwich structures. The black layers on the inner side of the wing are several layers of carbon fiber which decisively contribute to stiffness and strength of the wing structure. Only these carbon-fiber reinforced layers on the inner side of the wing are subject to optimization, whereas the number, the shape, and the orientation of the patches can be varied. The spar as well as the ribs remain unchanged, but this base structure is rather weak and without any reinforcement it would fail under maximum load.

7.5.2.2 Optimization model

Finite element model The geometry of the wing is modeled in CATIA V5 and imported into ANSYS as a surface model. The FE-mesh of the entire model is generated with layered shell elements (*SHELL91*) in order to make it compatible to FELyX. For the optimization process itself the graph-based representation is directly mapped to the FELyX FE-model which subsequently performs the FE-simulation. Due to the flat and nearly rectangular geometry of the wing structure a regular FE-mesh can be generated which is shown in Figure 7.11.

According to the flight manual of the DG-1000, the sailplane has to withstand a maximum acceleration of 6.4g. Assuming a total sailplane mass of 500kg, each wing has to provide a lifting force of $F_{tot} = 15.7$kN. The wing structure must sustain this load case at a temperature of $54°C$ without any damage at all.

[1] http://www.dg-flugzeugbau.de

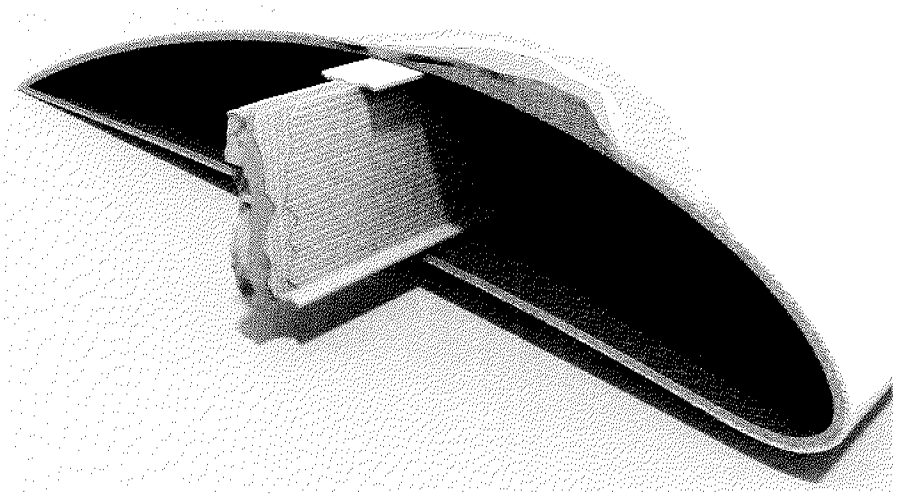


Figure 7.10: Detail of the wing structure.

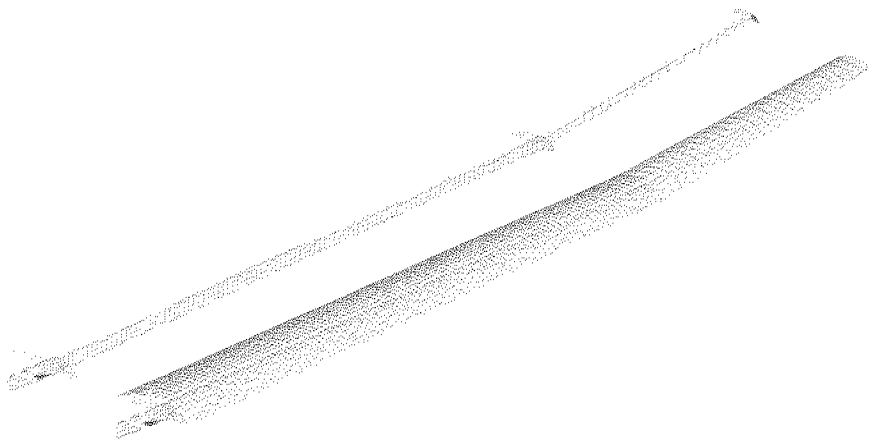


Figure 7.11: The FE-mesh of the spar and ribs (left image), and the FE-mesh of the entire wing (right image).

There are multiple possibilities to apply the aerodynamic loads on the wing structure. Probably the most precise way would be to perform a CFD-simulation to get a realistic load distribution. Another way is develop an analytic load model. The work of Howe [151] describes some approaches which can provide a rather realistic load distribution model. For all opti-

mization runs, an even more simplified approach is used. The total load of 15.7kN is equally distributed over the whole surface, whereas one third of the load acts on the under side and two thirds act on the upper side of the wing surface. For the optimization such a simple model is considered to be sufficient, but it is inevitable to validate the gained optimization results with a more detailed and accurate model.

The boundary conditions of the FE-model are also rather simple since the wing is hold by two simple bolts. Accordingly, the spar is fixed at two rows of FE-nodes in the region of the wing root as illustrated in Figure 7.12. Such a local fixation normally leads to stress and strain peaks, but since the local region around the fixation is not subject to optimization these peaks can be neglected.

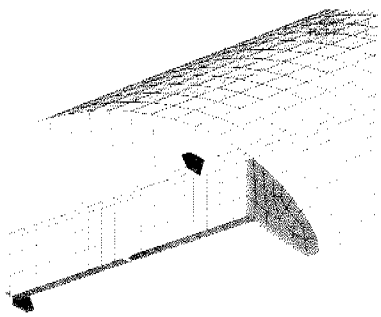


Figure 7.12: Boundary conditions of the FE-model.

Representation The wing is divided into eight sections along its length. Each of these sections is further divided into six zones, i.e., three zones on the upper and three zones on the under side of the wing. The zones above and below the spar are quite narrow, hence, additional straps can be applied to increase the stiffness of the wing. On the other hand, the zones covering the areas between the edges and the spar are kept rather large to keep the laminate structure simple. The only exception is the section at the wing root where the areas between the spar and the trailing edge are split again. In this section large forces occur and therefore local reinforcements can be useful to prevent the wing from failure. In Figure 7.13 the zone setup is depicted.

The graph-based representation is based on a total of 44 zones. These zones can be covered with three different types of carbon-fiber reinforced fabrics. HTUD is an unidirectional material which is characterized by a high ultimate tensile strength. HMUD, also an unidirectional fabric, is characterized by a high modulus of elasticity but its ultimate tensile strength is

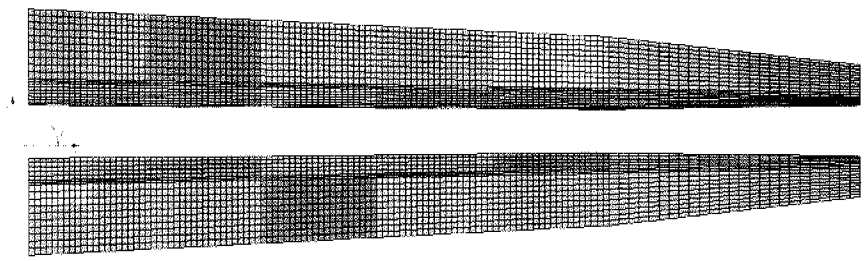


Figure 7.13: Zone-setup for the optimization. The upper image depicts the upper side of the wing while the lower image shows its under side. The orientation of the x-axis is the same for all elements and γ displays the angle definition of the laminate orientation.

reduced. Finally, a woven fabric with medium stiffness and strength in both material directions can be chosen.

The spar and the ribs are manufactured from glass fiber fabrics and foam. The basis of all optimization zones is a layer of foam and two layers of glass fiber which build a sandwich structure in combination with the additional carbon fibers. The parameters of all materials are listed in Table 7.2.

Material	HTUD	HMUD	weave	glass	foam	
E_1	135	220	55	44.5	0.15	[GPa]
E_2	10	7	55	13	0.15	[GPa]
ν	0.3	0.2	0.35	0.3	0.35	[-]
G_{12}	5	2.9	4.8	5	0.05	[GPa]
Thickness	0.15	0.15	0.3	0.3	6	[mm]
Area density	0.24	0.24	0.45	0.6	0.6	[kg/m^2]
X_t	1150	1100	615	1000	1	[MPa]
X_c	1400	1100	460	900	1.8	[MPa]
Y_t	55	50	460	50	1	[MPa]
Y_c	170	150	650	120	1.8	[MPa]
S	75	75	62	70	1.7	[MPa]

Table 7.2: Material data.

As can be seen from Table 7.2, the carbon-fiber fabrics are extremely thin. Consequently, numerous patches are required to prevent the structure from failure in particular at the wing root where the maximum bending moment occurs. If only thin layers are used, some zones require at least 20 patches in order to meet the failure criterion. Although the time to

evaluate the FE-simulation remains approximately the same, the overall optimization time is drastically increased due to the larger search space. Each patch vertex therefore holds an additional integer-gene which acts as a multiplier of the number of layers of the respective patch. Thus, the number of layers within a patch can be varied in discrete steps. This simplifies the optimization problem since mass reduction can be reached by reducing the overall patch thickness.

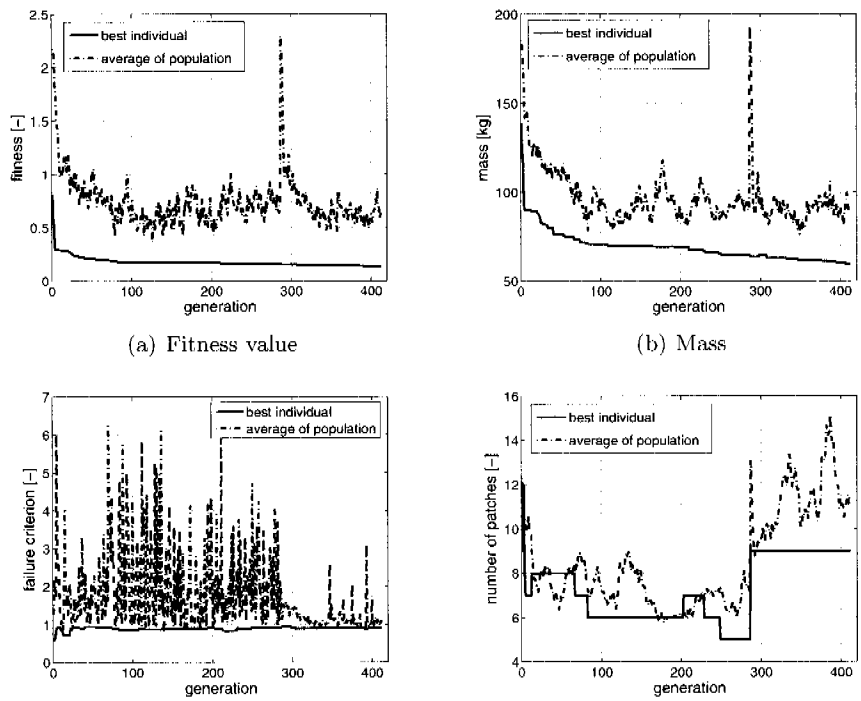
7.5.2.3 Optimization formulation

The objective is to reduce the mass of the wing structure in consideration of the following constraints:

- **Failure**: No element layers are allowed to have Tsai-Wu failure criterion values larger than 0.9.
- **Number of patches**: The number of patches has to be in between 5 and 30.
- **Empty zones**: All zones must be covered by at least one patch.

The minimum and maximum number of patches is a rather theoretical constraint serving the purpose of controlling the complexity of the optimized stacking sequence. In case the number of patches continually increases during the optimization and results in an optimum design with some dozens or even hundreds of patches, the manufacturing process would be too expensive. However, normally this constraint is inactive since the mass reduction objective typically leads to a relatively low number of patches. Furthermore, none of the zones should remain uncovered, i.e., empty zones are penalized with higher fitness values.

The more important constraint is the failure criterion. For composite laminates several failure criteria, e.g. Maximum Stress [152], Tsai-Wu [98], Tsai-Hill [153], or Hashin [154], exist which analyze the stress state in the structure. For this optimization the Tsai-Wu criterion is employed which is also outlined in Kress [155]. It combines the normal and the shear stresses to an index value which further depends on the maximum allowed tension (X_t, Y_t) and compression (X_c, Y_c) in both material principal directions as well as on the maximum allowed shear stresses (S). The critical parameter values of the materials at hand can also be found in Table 7.2. The criterion must be evaluated at each integration point of each layer of a finite element. Failure in an element occurs if the Tsai-Wu index equals or exceeds the critical value of 1 in at least one layer. The constraint which is considered during the optimization only allows for a failure criterion of 0.9 in order to include a relatively small margin of safety.



(a) Fitness value

(b) Mass

(c) Tsai-Wu failure criterion

(d) Number of patches

Figure 7.14: Fitness and objective plots for the wing optimization.

7.5.2.4 Results

The population size of the optimization is 60 individuals per generation. In the initialization stage 40 individuals are provided not violating the failure criterion and 20 completely random initialized individuals complete the first generation. A totally random initialized first population could hardly produce any feasible stacking sequences and therefore it might be computationally expensive to find a region of feasible solutions. A single evaluation takes about four minutes on a 2.8GHz Intel Xeon processor. A cluster of such processors is available and the average evaluation time for a generation is approximately a quarter of an hour. Consequently, the total number of evaluations is limited. Seeding feasible solutions into the first generation is definitely a reasonable way to speed up the optimization process, although the exploration of the entire search space is reduced.

Figure 7.14 shows the convergence behavior of the optimization run. After 286 generations the optimization seems to be converged. The solution is rather simple and consists of five patches only. Four patches cover the wing

root and the fifth patch covers the entire wing surface. The patch cannot be shrunk anymore because this would violate the rule that every zone must be covered by at least one patch. Decrementing the number of layers by one results in a failing structure and introducing a new patch increases the overall mass. Consequently, the algorithm is incapable of finding improved solutions.

A possible workaround is to split this patch into five patches, each having only a single layer. Thus, the optimization is continued after 286 generations whereas these patches can independently change their shape and position. This restart leads to peaks in the fitness and mass plots in Figure 7.14 which are caused by the restart itself. Only the current best individual is used for the restart and all the other individuals are randomly initialized in order to introduce new genetic information. Albeit not much, certain further improvements can be reached after a total of 412 generations.

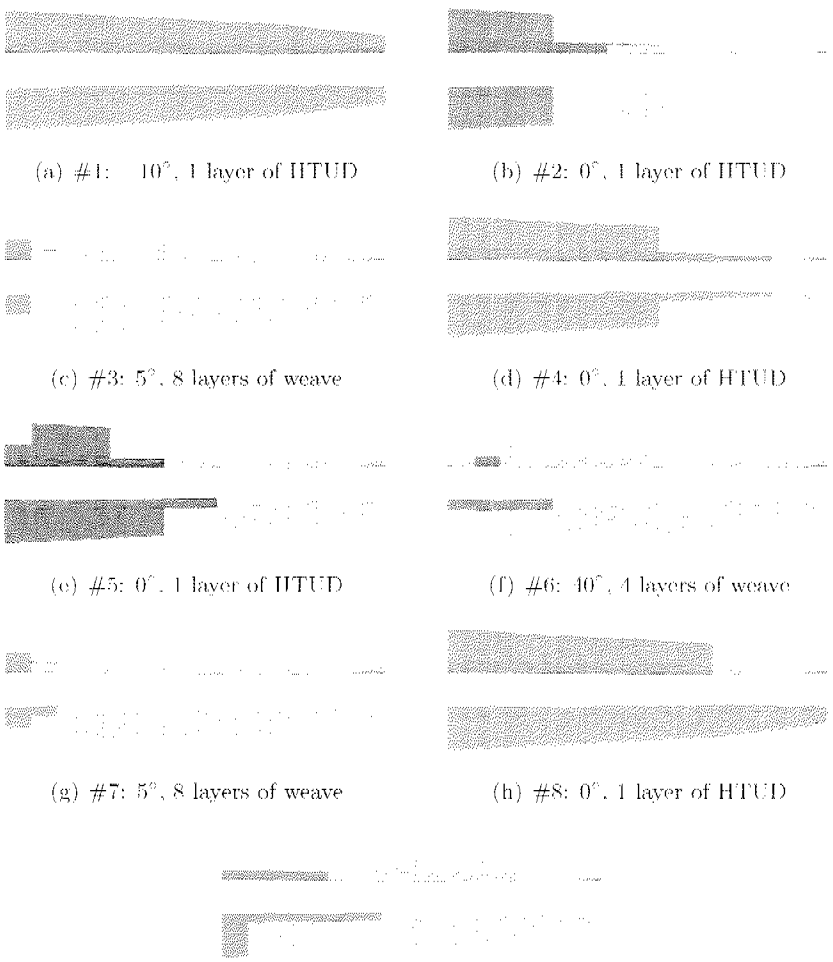
Finally, the best individual has a mass of only 59.2306kg without violating any constraint. The resulting patch shapes are shown in Figure 7.15. Only the first patch covers the entire wing structure and the constraint demanding for no uncovered zones is therefore met. All other patches cover mainly the wing root with HTUD fabrics, but also the weave material and the high-modulus fabrics are locally applied. Most of the patches are arranged on the zones on the leading edge and the zones behind the spar close to the trailing edge are only rarely covered.

This can be explained with the fact that the distance from the upper band of the spar to the lower band is shorter on the leading edge. Therefore a patch on the leading edge provides approximately the same stiffness but with a lower weight. In the critical zones near the root particularly the weave material with a rather high modulus of elasticity for both material directions is used. It is suited to distribute stresses on larger areas while the other materials only transfer forces in one direction.

7.6 Conclusion

A new graph-based representation concept for the evolutionary optimization of laminated composites structures is introduced. It is able to concurrently optimize size, shape, position, number, orientation, and any other material related property of fiber-reinforced patches. The definition of zones allows for an almost arbitrary inclusion of domain knowledge and the complexity of the optimization problem can be influenced independent from the granularity of the FE-mesh. The applications demonstrate the methods ability to find good quality results in constrained optimization problems.

However, the method could be further developed. Although the introduction of an integer-gene defining the number of layers of a patch can be advantageous to reduce the size of the search space, it might also reduce



(a) #1: 10°, 1 layer of HTUD

(b) #2: 0°, 1 layer of HTUD

(c) #3: 5°, 8 layers of weave

(d) #4: 0°, 1 layer of HTUD

(e) #5: 0°, 1 layer of HTUD

(f) #6: 40°, 4 layers of weave

(g) #7: 5°, 8 layers of weave

(h) #8: 0°, 1 layer of HTUD

(i) #9: 5°, 5 layers of HMUD

Figure 7.15: Resulting patches of the sailplane wing optimization.

the flexibility for local adaptations. A possible solution to overcome such problems as occurred in the sailplane wing optimization is the development of a *split* variation operator. This operator could split patches with several layers into independent patches of the same shape and position. Consequently, the fitness of the individual which is processed by the split operator is not changed, but further variation operator applications possibly profit from this operator since the newly created thin patches can be manipulated independently.

So far, the representation concept only works on simply connected areas.

As soon as at least three surfaces with coincident boundaries occur, single patches can be split and cover finite elements on all three surfaces what is impossible from a manufacturing point of view. A simple workaround is to optimize such surfaces independently, but then the cohesion of the structure is not guaranteed. For such highly-complex optimization domains novel approaches need to be found.

The sailplane wing application is a rather simple geometric form since it is predominantly flat and only single-curved. For applications with doubly-curved surfaces an additional issue needs to be considered. When draping weave materials on doubly-curved surfaces the relative orientation of the reinforcement fibers normally changes what finally leads to modified structural properties of the mechanical part. Such drapery problems are not yet discussed and require extensions of the method.

Chapter 8

Graph-based CAD optimization

8.1 Introduction

Modern CAD systems are essential for product development processes because they offer a parametric-associative representation of mechanical structures. These systems can handle extremely complex mechanical structures which are most often heavily constrained by functional and manufacturing demands. The parametric-associative representation concept is the basis for many gradient-based or stochastic optimization strategies which are available in several CAD packages, e.g. CATIA V5's Engineering Optimizer workbench or Unigraphics' Optimization Wizard[1].

In the dissertation of König [80] the EA-based parameter optimization of mechanical structures modeled in CATIA V5 is presented and Ledermann [156] performs a topology optimization of a wing structure with the optimization tool DynOPS [80]. Within the scope of this chapter König's approach is further developed to CAD-based topology and shape optimization. So far, only parameter variations of a given set of design parameters can be evaluated which are organized in a constant-length genotype. A novel graph-based representation concept is developed opening up the search space to a variable number of design entities by directly accessing the specification tree of a mechanical structure modeled in CATIA V5. A generic EC framework equipped with parallel computing capabilities is established which provides all required functionalities to optimize topology and shape of arbitrary mechanical structures.

In Section 8.2 the handling and data structure management of mechanical structures in CATIA V5 is discussed. The newly proposed specification-tree representation is described in Section 8.3 and the appropriate variation operators required for evolutionary optimization are the subject of

[1] http://www.ugs.com

Section 8.4. Some remarks on the implementation of the entire EC framework are given in Section 8.5. A simple minimum compliance optimization problem is examined in Section 8.6 for verification purposes and a more complex spar optimization problem is treated in Section 8.7 to demonstrate the capabilities of the novel CAD-tree representation concept.

8.2 Handling of mechanical structures in CATIA V5

The commercial CAD software CATIA V5 is chosen for the implementation of CAD-based topology and shape optimization because it is a sophisticated CAD software in terms of programming concepts and data structures. The C++ interface CAA V5 offers access to almost all data structures and functions what is an absolute prerequisite for the implementation of an EC framework directly manipulating mechanical structures modeled in CATIA V5. The following explanations are based on the manuals of CATIA V5 and on the documentation of its C++ interface CAA V5.

8.2.1 The containers of a part document

A complex mechanical structure, e.g. an entire car or an airplane, can be represented in a `Product` document in CATIA V5. A `Product` document assembles different components which can be `Product` documents themselves or which can be single mechanical structures represented in `Part` documents. Figure 8.1 shows an example `Part` document of a wrench with the corresponding specification tree holding a list of mechanical features. Such mechanical features can be understood as parameterized rules defining the appearance of geometrical or topological entities. The `Part` document holds four data containers which are strongly linked together:

- **Product container.** It manages the integration of a `Part` document into the `Product` document.
- **Specification container.** It contains the actual design representation of the mechanical object. The design is defined by a list of mechanical features which are hierarchically grouped in the specification tree. In general, three different kinds of mechanical features are represented in the specification tree:
  - The **Part** feature (`Wrench` in Figure 8.1) is the main feature containing the design of the object to be built.
  - The **geometrical feature sets.** They contain either other geometrical feature sets or geometrical features. In Figure 8.1 `PartBody`, `Add.Cipher`, `Body.Cipher`, `Geometrical.Set` and

`Body.Template.Cipher.2` belong to the group of geometrical feature sets.

- **Geometrical features.** These features hold a topological result. In Figure 8.1 several geometrical features, namely all pads `Pad.*` and their defining sketches `Sketch.*`, all fillets `EdgeFillet.*`, `Pocket.Shaft`, and all user-defined features `UDF.*`, are shown.

- **Scope container.** This container is concerned with generic naming concepts providing stable and flexible ways to reference topological geometry objects in the specification container.

- **Geometrical container.** Mechanical features handled in the specification container essentially capture the design intent of the user. From these features the actual shape of a mechanical object is computed using an underlying modeler, also called update mechanism, and the topological results are stored in the geometrical container.

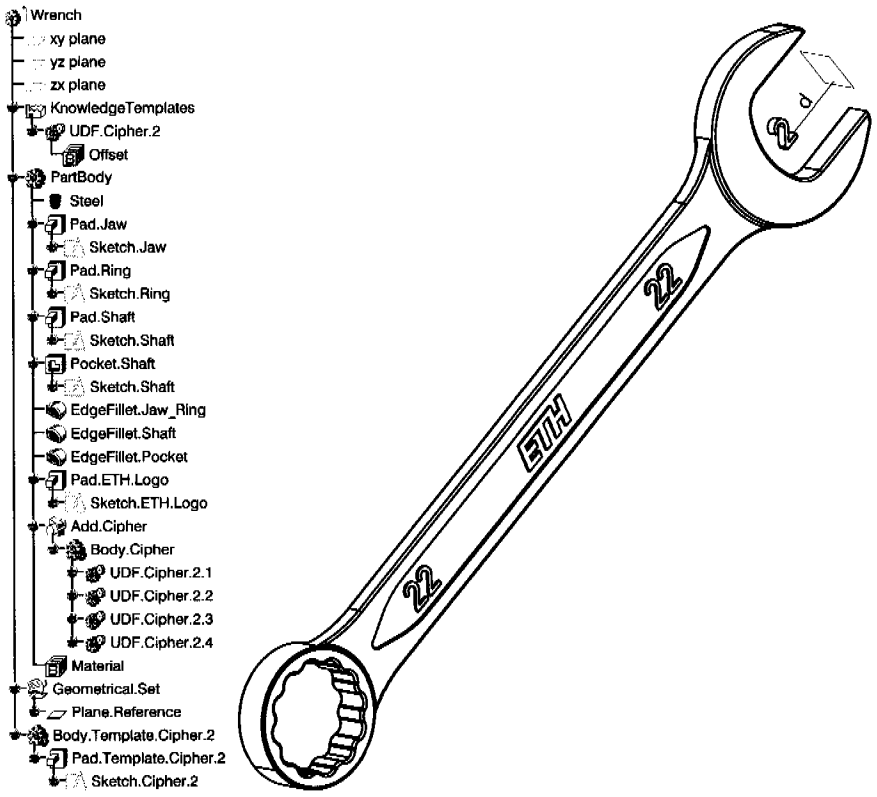


Figure 8.1: Specification tree and topological result of a wrench.

If a geometrical feature is modified, e.g. by altering a parameter value of a geometrical measure, the update mechanism processes the specification tree from this feature until the end of the specification tree is reached. Unfortunately, the update process cannot always be completed since inappropriate changes of geometrical features sometimes result in update errors. Typically, the update process fails due to geometrical constraints which cannot be met anymore or topological limitations which are violated. However, in case an update process is successful, the new topological result is displayed.

Next, a specialized feature available in the *Mechanical modeler* workbench is presented which is particularly suited for topology optimization of CAD structures.

8.2.2 User-defined features

Topology optimization is characterized by examining different solutions to a problem which have not necessarily the same number of geometrical entities. The user-defined feature (UDF) concept is ideally suited for topology optimization since it allows for the definition of template features which can be instantiated arbitrarily often.

An UDF is a set of features (geometric elements, formulas, constraints, etc.) which are grouped in order to be used in a different context as self-contained feature. The definition of UDFs is based on so-called *inputs*. These are features which are not part of the UDF, but external links of the UDF point at these input features. It is of high importance that these inputs are topologically stable, otherwise the instantiation of the UDF fails. Moreover, an UDF consists of *knowledge parameters* defining the underlying features. Some of these parameters can be published, i.e., these values can be modified during the instantiation step. The instantiated UDF therefore still meets the basic design intent but its appearance is appropriately modified. Furthermore, the UDF can be saved in a catalog and it can be reused in other `Part` documents.

In Figure 8.1 an illustrative example of an UDF is presented to clarify the concept. The cipher `2` is used four times in order to denote the size of the wrench. Instead of constructing four times the same cipher, a template of this cipher is modeled within a geometrical feature set (`Body.Template.Cipher.2`) which is not part of the `PartBody`. This template cipher is depicted inside of the wrench jaw and the published parameter d indicates the distance to the reference plane which is an input of this UDF. The UDF is shown in the specification tree under `KnowledgeTemplates` as `UDF.Cipher.2` with the published parameter d = `Offset` and its instances with different offset values are included in the geometrical feature set `Body.Cipher` which is added to the `PartBody`. The only published parameter is the offset d, but further parameters could be defined for altering the size or the position of the ciphers.

8.3 Specification-tree representation

The specification container holding the definition of geometrical features of a mechanical structure and the geometrical container holding the topological result are perfectly suited for evolutionary optimization. The specification container and its graphical representation, i.e. the specification tree, defines the history of the construction process of the mechanical structure and it can be considered as genotype. The topological result which is typically displayed as part is equivalent to the phenotype.

8.3.1 The graph-based genotype

The fundamental idea is to interpret the specification tree as mathematical graph and to apply the variation operators directly on this graph. Furthermore, the incorporation of UDFs leads to graphs and genotypes of varying size, respectively, which are suited for topology optimization[2]. Several requirements for the structure of the graph-based genotype can therefore be formulated:

- The genotype definition has to be generic to handle arbitrary mechanical parts.
- All parameters in the specification tree have to be accessible to optimization.
- An arbitrary number of UDFs of different type can be part of the specification tree for topology optimization.
- All standard geometrical feature sets and geometrical features have to be manageable.
- The graph-based representation has to be accessible to variation operators exclusively producing legal solutions.

In general, each geometrical feature set and all geometrical features need to be represented as vertices in the graph genotype and they have to be connected by edges according to the specification tree. As an example, Figure 8.2 depicts a half model of a bridge-like mechanical structure with its specification tree which will be optimized in Section 8.6. The corresponding graph genotype is shown in Figure 8.3, whereas only the `PartBody` is represented in the graph and the material definitions as well as any auxiliary constructions are omitted since they not directly influence the shape or the topology of the resulting mechanical structure. The specification tree is not

[2]In contrast to the previous chapter where composite laminates are optimized, this approach only works on solid geometries modeled in CATIA. Recent developments of CATIA would also render laminate optimization possible.

completely unfolded for better perceptibility, hence, a lot of parameters and constraints are hidden. Naturally, these parameters are also represented in the graph genotype to make the entire mechanical structure accessible to shape and topology optimization. Furthermore, the graph representation is not an exact copy of the specification tree since the mechanical features are not exactly accessible as it is displayed in the specification tree. As an example, `AbsoluteAxis` and `Geometry` are omitted from the graph genotype.

The graph genotype not only holds the connections between the mechanical features, but for all vertices some additional properties need to be stored. Three vertex properties are assigned to each vertex:

- **Name:** The name of the represented mechanical feature is stored as it is displayed in the specification tree.
- **Type:** For the build-up of the graph five types of mechanical features need to be distinguished.
  - *Feature:* Each mechanical feature is of this type in case not one of the following four types is assigned.
  - *OptFeature:* It represents a parameter which can be altered during optimization. Typically dimensions of a sketch are of this type.
  - *Container:* All geometrical feature sets are of this type in case they not include UDF instances.
  - *UDFcontainer:* This is a special container type which exclusively holds UDF instances of the same UDF template.
  - *UDFinstance:* All instances of an arbitrary UDF are of this type.
- **Vector of *UniGenes*:** Vertices of type *OptFeature*, *UDFcontainer*, and *UDFinstance* hold a vector of universal genes (cf. Appendix A). *OptFeatures* typically hold a single gene which defines the behavior of the respective parameter. The *UDFcontainer* requires an integer-gene defining the number of *UDFinstances* it contains and the gene vectors of *UDFinstances* are composed of several genes defining the appearance of the respective instance.

With these vertex properties the graph genotype defines the entire mechanical structure. In Figure 8.2 the types, and the gene definitions are depicted in italic besides the respective names of the mechanical features.

8.3.2 Modeling rules

In CATIA V5 mechanical structures can be constructed in various ways. However, it is mandatory to follow some modeling rules to create a stable optimization model, but none of these rules generally limits the modeling possibilities. The rules which are also applied in Figure 8.2 are the following:

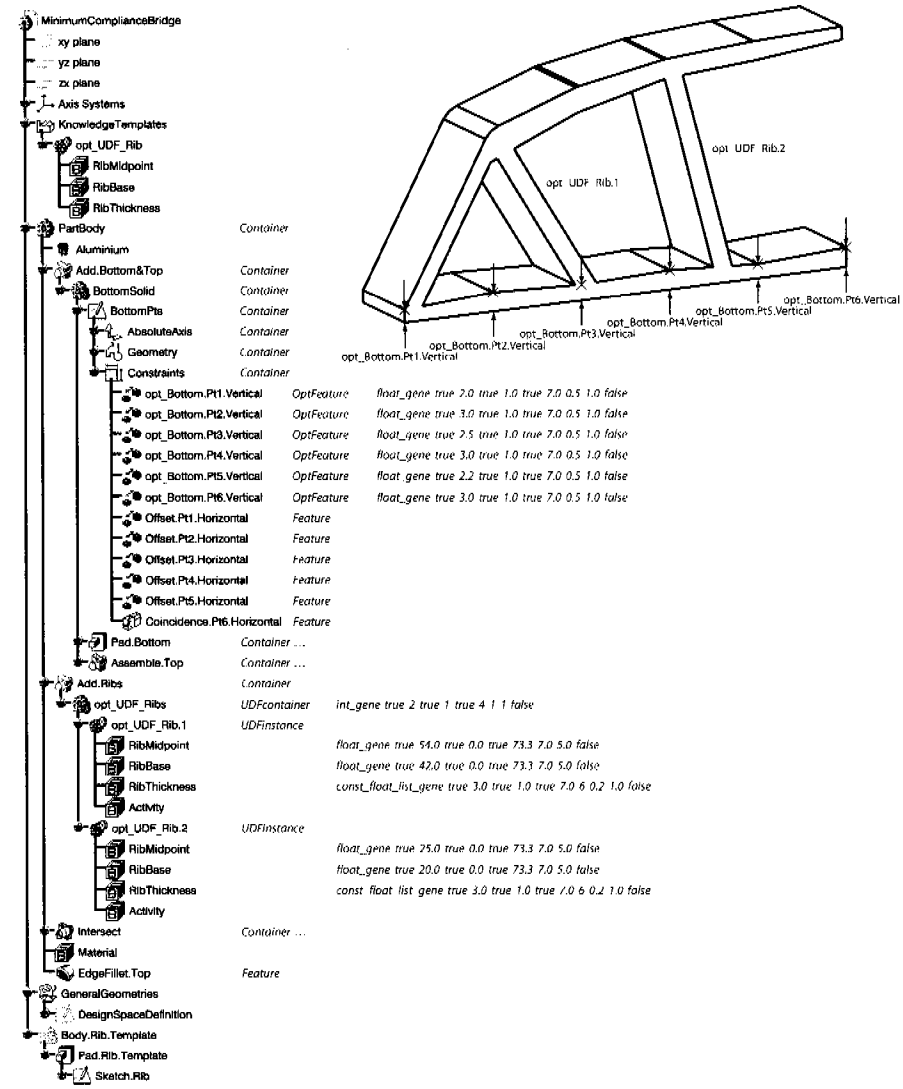


Figure 8.2: Bridge-like mechanical structure. The vertex properties are displayed besides the respective mechanical features in italic.

- The names of all *OptFeatures* are labeled with the prefix 'opt_' and the names of *UDFcontainers* and *UDFinstances* are marked with the prefix 'opt_UDF_'.
- *UDFcontainers* exclusively contain *UDFinstances* which are derived from the same UDF template. Other features are not allowed to be in

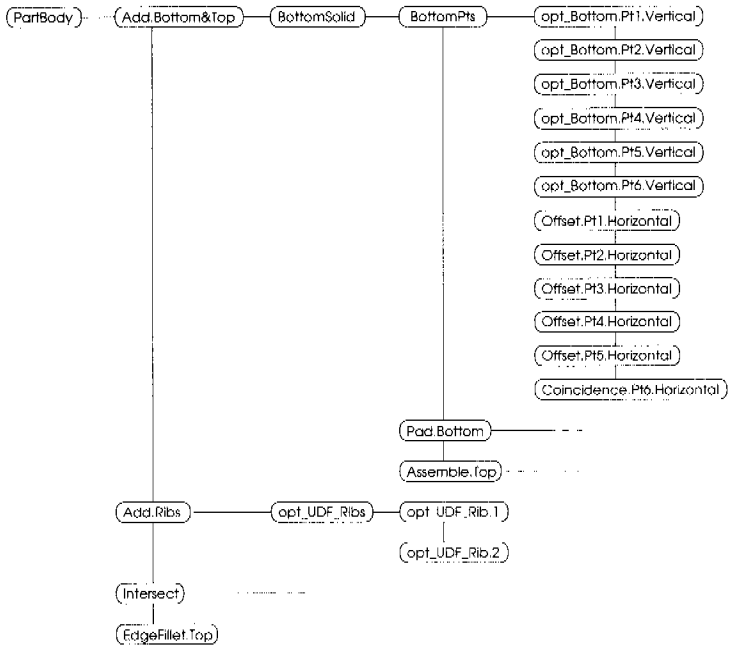


Figure 8.3: Graph representation of the specification tree of the bridge-like mechanical structure.

UDFcontainers.

- The name of *UDFcontainers* has to completely contain the name of the corresponding UDF templates, e.g., the container holding instances of the *opt_UDF_Rib* template are named *opt_UDF_Ribs* but also *opt_UDF_Rib_container* would be an alternative.

8.4 Variation operators

Two different types of variation operators are required for the above presented graph representation. The first operator type only addresses *Opt-Features* which are always at the same position in the graph genotype and normally lead to shape variations. The second operator type handles *UDFinstances* which are contained in *UDFcontainers*. Not only the parameters defining the appearance of the *UDFinstances* are modified, but also the number of these instances incorporated within an *UDFcontainer* is altered. Thus, these variations operators modify the shape as well as the topology of the mechanical structure.

8.4.1 *OptFeature* variation operators

The mutation and crossover operators working on the *OptFeatures* are the standard variation operators known from the traditional constant-length representations. Since the genes are implemented according to the definitions given in Appendix A, the uniform as well as the Gaussian mutation operators modify the *OptFeature* parameter values with a predefined probability. Furthermore, four different crossover operators are applied, namely one-point, uniform, segment, and hypercube crossover as introduced in Section 2.4.2. Their implementation is straightforward because each *OptFeature* from the first parent has a well-defined counterpart in the second parent independent from the variable-length *UDFcontainers*.

8.4.2 *UDFcontainer* and *UDFinstance* variation operators

These variation operators only act on *UDFcontainers* and their *UDFinstances*. In general, they have to cope with an unequal number of *UDFinstances* contained in the *UDFcontainers*. According to the modeling rules an *UDFcontainer* can exclusively hold *UDFinstances* which are derived from the same UDF template. This organization of the mechanical features decisively simplifies the application of the crossover operators since each *UDFcontainer* of the first parent has a distinct counterpart in the second parent.

8.4.2.1 Mutation

On the gene level the uniform and Gaussian mutation operators work analogously to the mutation operators acting on the *OptFeatures*, i.e., the gene values stored in the vertex properties of all *UDFinstances* are altered with a predefined probability.

Furthermore, the number of *UDFinstances* contained in a *UDFcontainer* is subject to uniform-instance and Gaussian-instance mutation. If the number of instances is reduced, randomly chosen instances are deleted. On the other hand, an increasing number of instances produces additional randomly initialized *UDFinstances*. The integer-gene stored in the vertex property of the corresponding *UDFcontainer* defines the lower and upper limit for the number of instances.

8.4.2.2 Crossover

The exchange-instance crossover exchanges all *UDFinstances* of corresponding *UDFcontainers* of both parents with a predefined probability. Further variable-length crossovers are the uniform-instance, the segment-instance, and the hypercube-instance crossover. They work analogously to the standard implementations presented in Section 2.4.2, but they are able to handle a variable number of instances. If the *UDFcontainers* of the parents are of

different size, either the instances with identical distances from the container vertex are processed or a proportional strategy is applied. The proportional strategy is based on the numbers (np_1, np_2) of *UDFinstances* of both parent *UDFcontainers*. Each *UDFinstance* of the smaller *UDFcontainer* (index 1) is processed and the counterpart of the second parent (index 2) is determined by the following rule:

$$p_2 = \left\lfloor \frac{np_2}{np_1} p_1 \right\rfloor, \tag{8.1}$$

where p_1 and p_2 denote the number of edges in between the *UDFinstance* vertex and the respective *UDFcontainer* vertex. Finally, a one-point-instance crossover is implemented which determines crossover points either equidistant from the *UDFcontainer* vertices or with a proportional strategy similar to the variable-length one-point crossover described by Equation 5.2.

8.5 Implementation

Additional vertex and edge properties are introduced in order to simplify the processing of the graph genotype in the mapping stage. Each vertex holds a unique label, the number of direct children as well as an integer number indicating its level, i.e., the `PartBody` which is always the first vertex has level 0 and the following vertices have incremented level values according to their depth in the graph. Also the edges are assigned with a level property indicating whether two connected vertices are on the same level or not. With these additional information the graph processing is straightforward.

The C++ interface CAA V5 is used for all implementations concerning the manipulations of the specification tree and the FE-simulation of the updated solutions. The basic EA functionalities are provided by EOlib [93] and the handling of the mathematical graphs is based on BGL [139]. Additionally, the UniGene concept (cf. Appendix A) provides the different gene types required for the evolutionary optimization.

The setup of the optimization of a mechanical structure is carried out in two steps. First, an initializer program is run which simply processes the specification tree and writes a text file containing the respective graph in a predefined formatting. The parameters which are labeled with the prefixes 'opt_' or 'opt_UDF_' are recognized and a default gene definition is added to the text file. These gene definitions, i.e. lower and upper limits as well as mutation parameters, need to be adjusted before the optimization process can be started.

The computational costs are still a critical issue of evolutionary optimization. A single FE-simulation required to estimate the quality of a design solution often takes several minutes, hence, the evaluation stage is the most expensive step of the evolutionary process. König's [80] EA-based parameter optimization is directly implemented in CATIA V5 without any

parallel computing capabilities. Thus, the complexity of the model to be optimized is limited in order to get optimization results in reasonable time. It is therefore extremely important to develop an EC framework which allows for parallel computing. The architecture of the EC framework presented in this chapter is equipped with parallel computing capabilities provided by PVM [99]. The master process handles all stages of the evolutionary process (cf. Figure 2.3) except for the evaluation step and the FE-simulations are performed by slave processes which are distributed to several cluster nodes.

8.6 Minimum compliance problem

8.6.1 Problem description

A bridge-like minimum compliance problem is investigated to test the graph-based representation concept. In Figure 8.4 the setup of the optimization problem is illustrated, whereas the design domain is indicated by dashed lines. The structure consists of a lower and an upper plate and a varying number of ribs which are placed in between. It is simply supported on the left side, whereas a symmetry boundary condition is applied on the right side and a surface force acts on the bottom surface.

The optimization objective is to minimize the compliance of the structure. The elastic energy is proportional to the compliance of a structure and can therefore be taken as a measure for the design objective, hence, the elastic energy has to be minimized. Furthermore, a mass constraint is defined which limits the structure to fill 20% of the entire design domain. According to the specifications in Table 8.1 the mass of the structure is restricted to 0.02397kg.

8.6.2 Representation

The representation of the structure is shown in Figure 8.5. The lower plate has a planar functional face at the bottom and its upper side is defined by six horizontally equidistant points. Their vertical components $t_{b1} \dots t_{b6}$

Parameter	Denotation	Value	Unit
Dimensions	$l \times w \times h$	$73.7 \times 15 \times 40$	$[mm]$
Surface load	f	14.4	$[MPa]$
Modulus of elasticity	E	70	$[GPa]$
Material density	ρ	2710	$[kg/m^3]$
Target mass fraction	m_t	20	[%]

Table 8.1: Specifications of the minimum compliance problem.

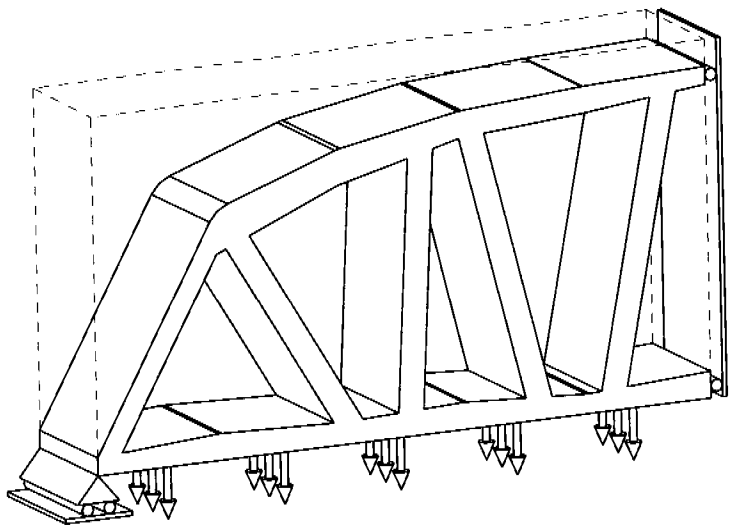


Figure 8.4: The design space and the boundary conditions of the minimum compliance problem.

are ordinary optimization variables defining the upper contour of the lower plate. The upper plate is defined by two sets of sampling points, i.e. six horizontally equidistant height parameters $h_{u1} \dots h_{u6}$ defining the top face and six thickness variables $t_{u1} \dots t_{u6}$ defining the bottom face of the upper plate. On the left side of the design domain where the structure is simply supported the representation always connects the lower and the upper plate. This is definitely reasonable from an engineering point of view and can be considered as inclusion of domain knowledge. Finally, the edges of all faces are chamfered in order to smooth the contours.

The ribs, only one is shown in Figure 8.5, are defined by three optimization variables, namely a lower position x_{bi}, a middle position x_{mi}, and a thickness measure x_{ti}. In Figure 8.2 an example of the structure with two ribs is shown, but for the optimization a minimum number of one and a maximum number of four ribs are defined. For all optimization variables lower and upper limit values are listed in Table 8.2.

8.6.3 Evaluation

The fitness formulations introduced in Section 2.5 are employed for the determination of the fitness values. The mapping function for the design objective, i.e. the elastic energy, is defined by $O_{init} = 0.3\text{J}$ and $O_{estim} = 0.0\text{J}$. An adaptive constraint function penalizes constraint violating individuals, whereas this function is defined by $C_{limit} = 0.024\text{kg}$ and $C_{feas\_tol} = 0.005\text{kg}$.

The FE-simulation and the evaluation of elastic energy and mass mea-

Parameters	Range [mm]
t_{b1}	$[0.0, 10.0]$
$t_{b2}...t_{b6}$	$[0.0, 8.0]$
h_{u1}	$[0.0, 10.0]$
$h_{u2}...h_{u6}$	$[0.0, 30.0]$
l_{u1}	$[0.0, 10.0]$
$t_{u2}...t_{u6}$	$[0.0, 8.0]$
x_{li}	$[0.0, 73.7]$
x_{ui}	$[0.0, 73.7]$
x_{ti}	$[0.5, 8.0]$

Table 8.2: Ranges of the optimization variables.

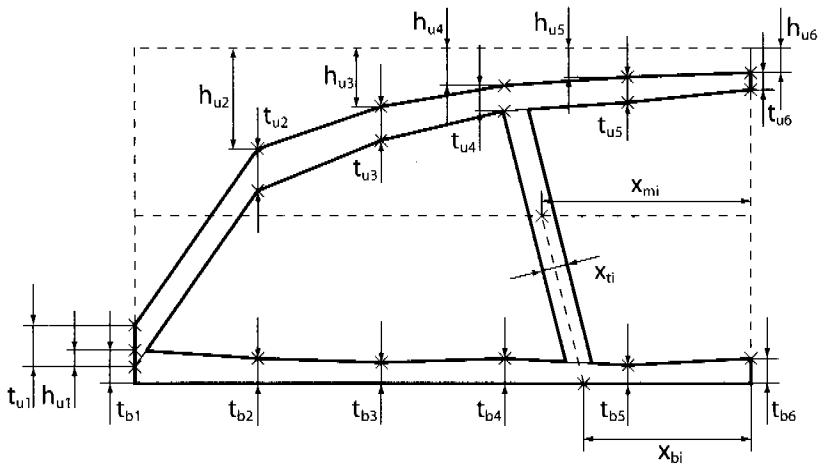


Figure 8.5: Representation of the bridge-like mechanical structure.

sures are performed in CATIA V5. The built-in FE-workbench GPS (Generative Part Structural Analysis) provides all required functionalities to execute FE-analysis and so-called sensors yield the required result data. The structures with altered topology and shape are remeshed for each evaluation with a constant mesh size. In case the update process fails, i.e., a combination of optimization variables leads to an illegal solution, an error fitness value is assigned to the respective individual.

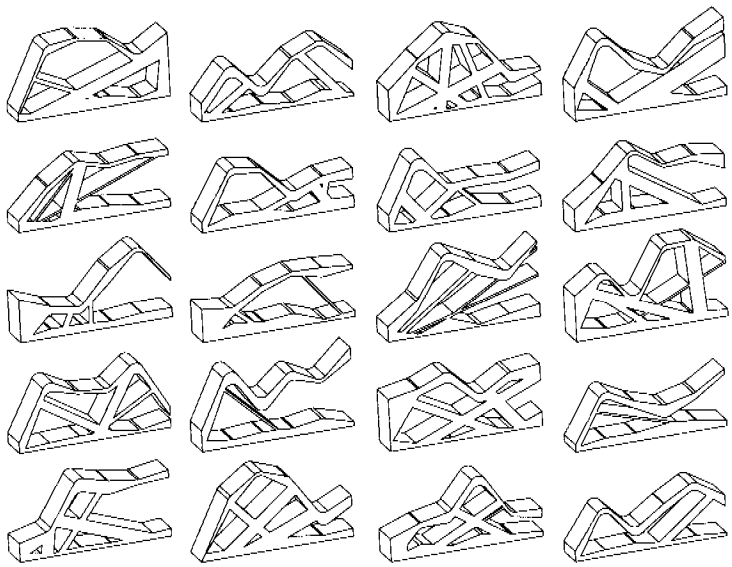


Figure 8.6: Selection of randomly initialized individuals of the minimum compliance problem.

8.6.4 Results

Several optimization runs are performed all having a population size of 60 individuals which are completely random initialized. A selection of an arbitrary initialization population is depicted in Figure 8.6. It is important to note that the number of ribs varies in between one and four, i.e., the initialization population incorporates topologically different individuals.

The different optimization runs generally produce optimum solutions with similar fitness values, but the number of ribs varies in between two and four. Thus, the optimization problem is assumed to be non-convex and several runs get stuck close to local optima. Unfortunately, it is impossible to interpret the optimization performance, since this kind of optimization problem cannot be solved by other optimization algorithms included in CATIA V5.

Nevertheless, the obtained results can at least be compared to the optimization results of König [80] who investigates an almost identical minimum compliance problem. The design domain and the loads are identical, but the representation slightly differs in the region of the simply supported edge and the minimum thickness of the ribs is 1mm compared to 0.5mm as shown in Table 8.2. Furthermore, the parameters of the fitness functions are unequal and the adaptive constraint function is not used by König. A constant-length vector genotype is used as representation, i.e., the number of ribs is set to four for the entire optimization. The best solution ever found by

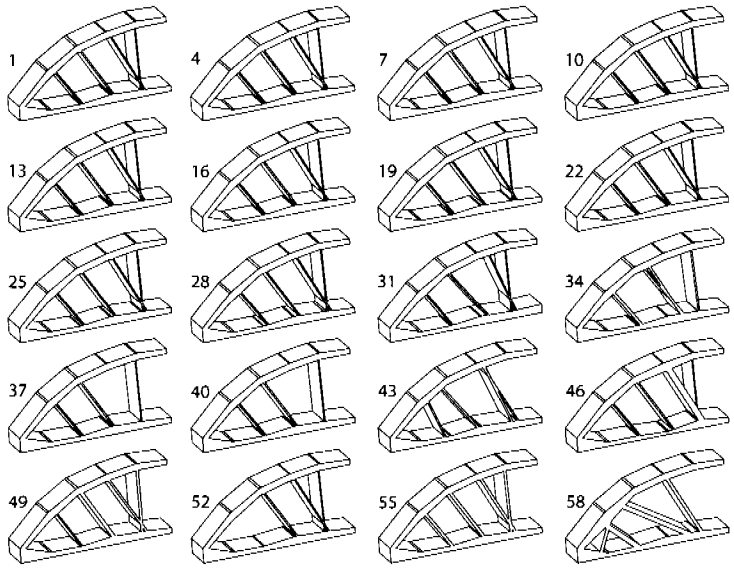


Figure 8.7: Selection of individuals of the final population. Each third individual is depicted from top left to bottom right.

König weights 0.0243kg, i.e. the mass constraint is slightly violated, and the elastic energy evaluates to 0.0944J. It is found after 120 generations with an identical population size of 60 individuals.

Although both optimization setups are not exactly equal, the graph-based representation concept outperforms the solution of König. The best solution found after 120 generations weights 0.0234kg and the elastic energy evaluates to 0.0732J. The optimization process is continued until 250 generations are completed, but the objective (0.0721J) and constraint (0.0233kg) values are only slightly improved. In Figure 8.7 a sorted variety of individuals of generation 250 are shown. A lot of individuals are similar indicating a converged optimization process, but also individuals with varying topology, e.g. individuals 37 and 40, are hold in the population. The best individual (number 1) consists of the maximum number of four ribs, whereas three of them are almost parallel. The fourth rib intersects with another rib what is probably suboptimal from a manufacturing point of view, but from a mechanical point of view such an intersection can be absolutely useful.

8.7 Spar optimization

The graph-based representation concept is ideally suited to optimize structures which can be constructed with a varying number of design entities. After the simple minimum compliance problem treated above a more com-

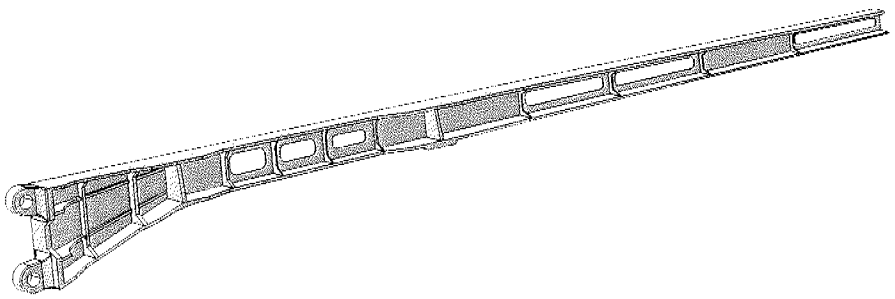


Figure 8.8: Arbitrary configuration of the spar.

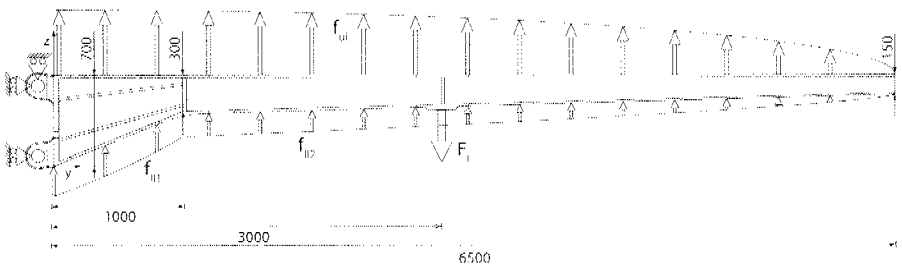


Figure 8.9: Load introduction of the spar model.

plex structural part is considered, namely the spar of an airplane wing. In Figure 8.8 a possible configuration of a fictitious spar model is shown which is constructed for the purpose of this optimization. It is not part of an existing airplane wing and further details of ordinary spars are omitted in the presented CAD-model. Moreover, it is assumed that the spar is integrally milled, i.e., the spar consists of one structural part and no additional assembly steps are required.

8.7.1 Problem description

The spar consists of three wing boxes, namely the inner, the middle, and the outer wing box as shown in Figure 8.11, and the wing span amounts to 6500mm. A maximum acceleration of 8g is assumed to be the critical load case. The mass of the airplane should be approximately 2800kg and each spar carries an additional wing tank payload of 200kg which is fixed at a distance of 3000mm from the wing root. The total mass is scaled by factor 2 for safety reasons and a single wing has therefore to produce a lifting force of 256kN.

The load introduction is subdivided into two surface loads which are applied on the lower and the upper caps of the spar. Typically, the maximum

lifting force occurs at the wing root and follows an elliptical distribution which is reduced to zero at the wing tip. Furthermore, it is assumed that $\frac{2}{3}$ of the lifting force act on the upper cap ($f_{ul} = 170.7$kN) and the remaining $\frac{1}{3}$ is applied on the lower cap ($f_{ll1} + f_{ll2} = 85.3$kN). The additional force $F_L = 32$kN caused by the payload is applied as surface load as illustrated in Figure 8.9.

The boundary conditions are required to produce a statically determinate FE-model. The upper flange at the wing root is simply supported in x-, y-, and z-direction, whereas the lower flange is only blocked in y-direction in order to prevent the spar from rotating around the upper flanch.

The objective is to reduce the mass of the spar in consideration of the following constraints:

- **Stress**: The maximum von Mises stress is not allowed to exceed a threshold value of 230MPa.
- **Stability**: A buckling analysis is run for each evaluation and the spar must not fail due to structural instability.
- **Stiffness**: The maximum rotation about the y-axis at the wing tip must not exceed 5°.

The spar should be manufactured from the standard aluminum alloy AA6061-T6. The most important physical and mechanical properties are given in Table 8.3. For the critical load case the tensile yield strength must not be reached with an additional safety factor of 1.2 what leads to a threshold stress value of 230MPa.

The structural stability of the spar is tested by running a buckling analysis. Such analyses are provided by the GPS workbench of CATIA V5 and they produce so-called load factors as analysis result. Load factors smaller than 1.0 indicate a local or global structural instability of the structure, whereas values above the threshold value of 1.0 guarantee for structural stability.

Density	2700	[kg/m^3]
Modulus of elasticity	68.9	[GPa]
Poisson's Ratio	0.33	[-]
Tensile yield strength	276	[MPa]
Ultimate tensile strength	310	[Mpa]

Table 8.3: Mechanical and physical properties of the aluminum alloy AA6061-T6.

	Mass	Stress	Stability	Stiffness
O_{init}	300.0 [kg]			
O_{estim}	200.0 [kg]			
C_{limit}		230 [MPa]	1.2 [-]	5.0 [°]
C_{feas_tol}		12.5 [MPa]	-0.1 [-]	2.5 [°]
C_{feas_init}	400.0 [kg]	1300 [MPa]	0.2 [-]	35.0 [°]

Table 8.4: Objective and constraint definitions for the spar optimization.

The third constraint is a torsion stiffness value measuring the rotation about the y-direction at the wing tip. This rotation is restricted to 5° in order to prevent too large deformations which could lead to failure of the wing surface or other adjacent structural parts.

The fitness formulation from Section 2.5 is employed and the respective values are stated in Table 8.4. The already known parameters are enhanced with a further parameter C_{feas_init} which is required for the initialization stage of the optimization. The randomly initialized individuals are only added to the first generation if these maximum values are not exceeded for the design objective as well as for the constraints and the initialization process is continued until a predefined number of feasible individuals is produced. Thus, the actual optimization process is started with qualitatively acceptable solutions and in particular no individuals failing during the update process belong to the initial population.

8.7.2 Representation

The parametric-associative CAD-model of the spar is defined by numerous parameters. Some of these parameters are used as ordinary optimization variables changing the shape of the spar structure. In Figure 8.10 a selection of these optimization variables is shown. Their lower and upper limits need to be carefully defined, otherwise an update after changed optimization variable values is likely to fail. In case of a random initialization this would lead to a high amount of failing individuals what would be a bad basis for an optimization run.

A varying number of ribs are arranged within the three wing boxes. This is realized by modeling three template ribs which are the basis of UDFs whose published parameters are the thickness and the position of the lower and the upper rib point. In Figure 8.11 template ribs for each wing box are shown. The distance of the upper point of each rib from the left box boundary is defined relatively to the length of the respective wing box, i.e., values in the range [0,1] are required as optimization variables. The lower point of each rib is constrained to move on the lower box boundary and

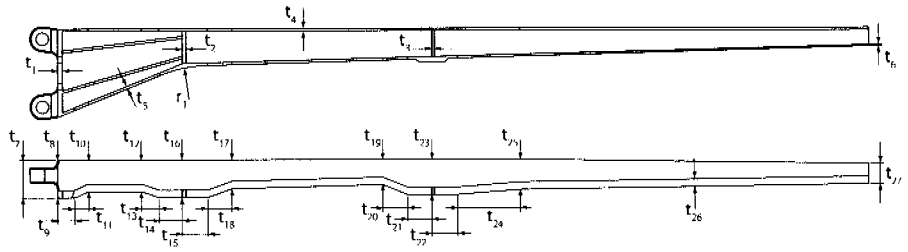


Figure 8.10: Ordinary optimization variables.

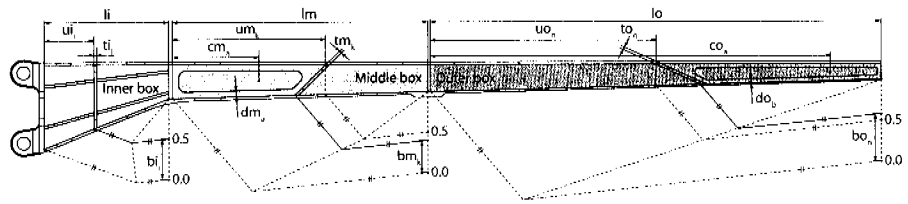


Figure 8.11: Representation of spar ribs and cutouts.

also a relative measure in the range [0,1] is used to determine the exact position of the point. A value of 0.5 leads to a rib which is exactly vertical, hence, deactivating this optimization variable at the default value of 0.5 provides optimization results only containing vertical ribs. The auxiliary constructions shown in Figure 8.11 are also part of the respective UDFs and they are updated depending on the position of the upper point.

For a further mass reduction cutouts can be placed within the middle and the outer box. These cutouts are also defined as UDFs and the number of them can be changed during the optimization process. They are defined by two parameters, namely the position of the cutout and the distance between the boundary of the cutout and the lower and upper box boundaries as well as the adjacent ribs. The position is also given as a relative measure depending on the length of the respective wing box, whereas this position only defines in between which ribs the cutout is placed but has no further influence on the shape of the cutout. Consequently, the mapping of the continuous optimization variable is heavily non-injective, since all the cutouts with position variables between the same pair of ribs are placed above each other. Such an UDF is highly dependent on the topology of the respective wing boxes what hardly can be achieved by other modeling strategies than parametric-associative CAD techniques.

The original shape of the ribs is limited by the web and the lower and upper caps, hence, they are of quadrangular shape. Further mass reduction

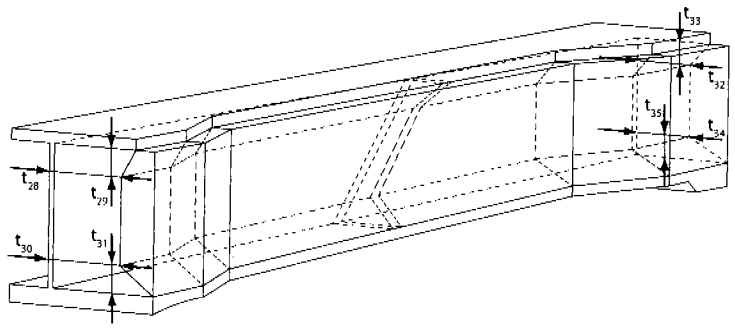


Figure 8.12: Rib modification of the middle wing box.

can be achieved by subtracting an extruded body from the ribs which follows the contour of the lower and upper caps. In Figure 8.12 the subtraction is illustrated for the middle wing box resulting in hexagonal ribs. Alternatively, the UDFs of the ribs could be parameterized as a hexagon, but this would lead to even more optimization variables.

8.7.3 Results

8.7.3.1 Vertical ribs

The optimization is run with a population size of 120 individuals and during the initialization stage 80% feasible individuals have to be generated. From a mechanical point of view it is not absolutely necessary to allow for inclined ribs for the critical load case presented in Section 8.7.1. Thus, the optimization variables determining the lower position of the ribs are deactivated and set to a constant value of 0.5.

The optimization result is depicted in Figure 8.13 and the von Mises stresses are shown in Figure 8.14. The final result consists of a varying number of ribs in the different wing boxes. In the inner wing box only two ribs are required reinforcing the boundary to the middle wing box. Three ribs are placed in the left half of the middle wing box which help to fulfill the torsion constraint and prevent from buckling of the caps. Finally, the outer wing box contains five ribs and four cutouts are placed in between for saving weight of the structure.

The convergence plots in Figure 8.15 show the fitness, the objective, and the constraint values of the best individual and the average of the respective generation. The convergence behavior of the fitness values plotted in Figure 8.15(a) shows some steps which are caused by the adaptive constraint definitions. In the beginning of the optimization process only solutions violating the maximum von Mises stress constraint are found and therefore the constraint function definition is adjusted in order to increase the penal-

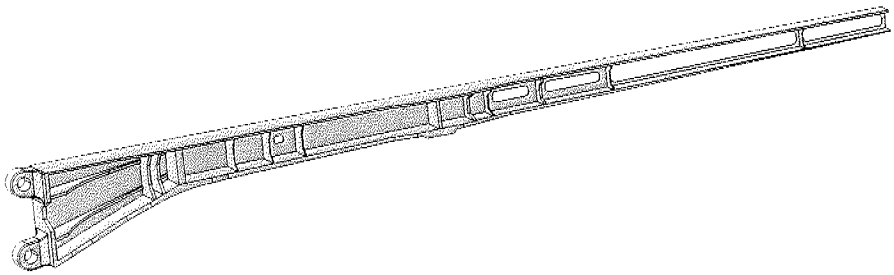


Figure 8.13: Result of the spar optimization with vertical ribs.

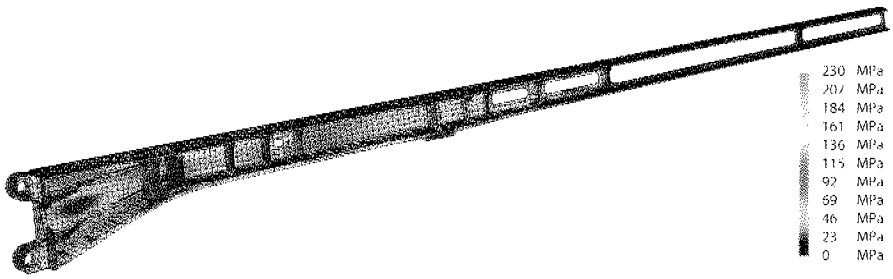


Figure 8.14: Von Mises stresses of the optimization result with vertical ribs.

ization of high stresses. During the first hundred generations the mass of the individuals is continuously increased in order to reduce the maximum stresses and to find solutions fulfilling the torsion stiffness constraint. In particular small cutouts lead to high stresses, hence, only few cutouts remain in the genotype of the best solutions. The buckling constraint is not critical at all since the thicknesses of the web as well as of the caps are rather high. Once all these constraints are met, the optimization process starts reducing the mass of the spar structure until the optimization process is stopped after 339 generations and finds a solution weighting 264.205kg. In contrast to the beginning of the optimization process the number of cutouts and their sizes are increased what leads to a substantial reduction of mass. A further improvement is unlikely with unchanged strategy parameters, but a restart of the optimization process with adjusted operator rates or changed fitness function definitions may lead to even better results.

8.7.3.2 Inclined ribs

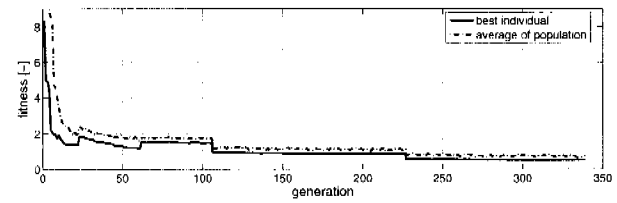
The result of the optimization with exclusively vertical ribs leads to a rather ordinary solution. A further optimization run is performed allowing for

inclined ribs as introduced in Section 8.7.2 in order to tap the full potential of mass reduction. The optimization variables determining the lower positions of the ribs are activated and their values are restricted to a range [0.1, 0.9]. Therefore, the shape of the cutouts is generally more complex since they depend on the positioning of the adjacent ribs. The given constraints are unchanged and all the strategy parameters of the optimization process remain the same as for the above presented optimization.

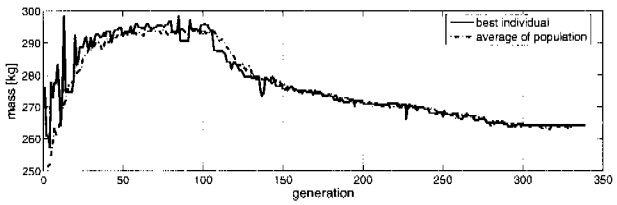
The optimization result is depicted in Figure 8.16 and the von Mises stresses are shown in Figure 8.17. In the inner wing box the topology resembles the above presented solution with vertical ribs although an additional third rib is placed close to the right box boundary. These ribs reinforce the region where the maximum bending stresses occur what additionally leads to a stiffening of the common boundary of the inner and the middle wing box. Three ribs are placed in the middle wing box, but they are differently arranged compared to the solution with vertical ribs. They are positioned in the right half of the box and their topology resembles a truss structure and two additional cutouts lead to a further mass reduction. Finally, within the outer wing box only two ribs and a single cutout are placed what is a reduction of complexity compared to the solution with vertical ribs.

During the optimization the rotation stiffness constraint as well as the von Mises stress constraint are active, whereas the latter is more difficult to be fulfilled. The application of the adaptive fitness definitions is of great value since the adaptation emphasizes the importance of the initially violated stress constraint. Thus, the mass increases in the beginning of the optimization process until solutions with rather low stress values close to the threshold value are found. Only the stability constraint is uncritical since the threshold value is not reached during the optimization process. The final mass after 229 generations is 237.659kg what is approximately 10% less than the result of the optimization with vertical ribs. A further improvement is still possible, but the expensive evaluation of individuals limits the number of generations.

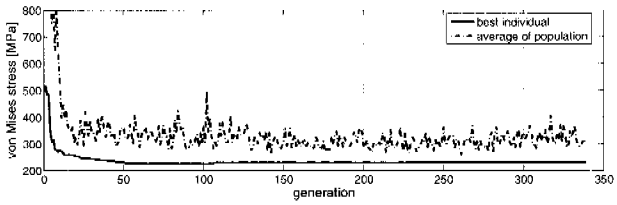
It is important to note that such a spar structure would have to fulfill many more constraints and several further load cases would have to be simulated. The purpose of this optimization is only to demonstrate the application of the graph-based CAD representation. The construction of a real spar structure would be much more expensive and can therefore not be treated within this thesis.



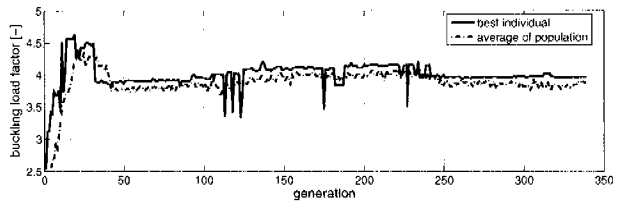
(a) fitness



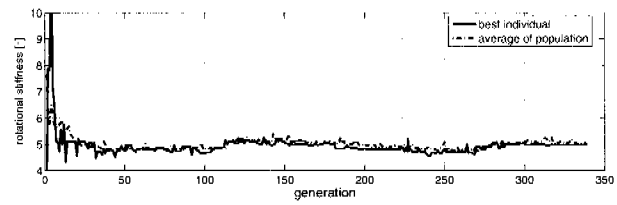
(b) mass



(c) stress



(d) stability



(e) stiffness

Figure 8.15: Convergence plots of the spar optimization with vertical ribs.

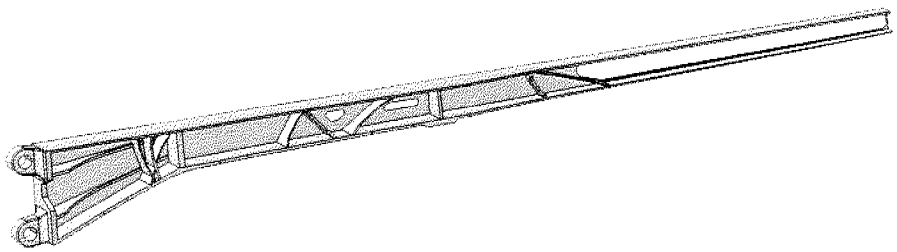


Figure 8.16: Result of the spar optimization with inclined ribs.

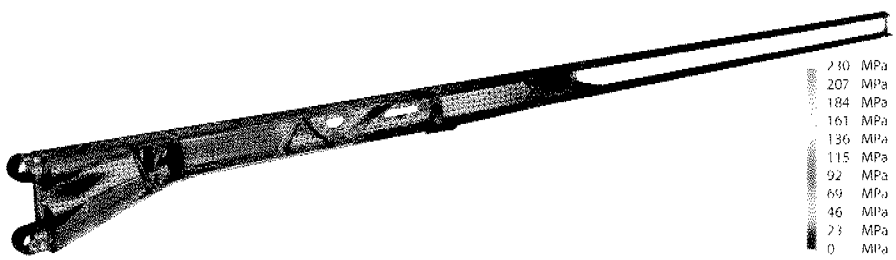
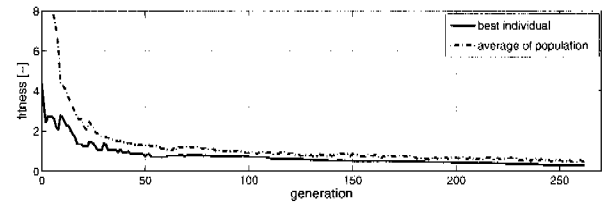
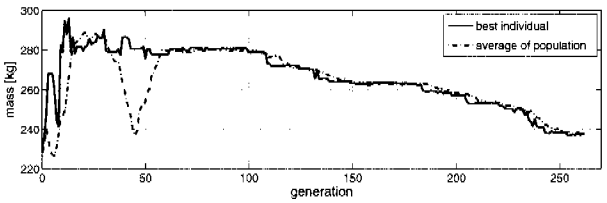


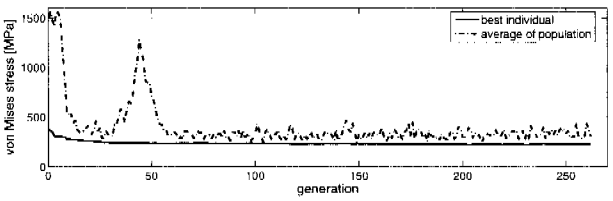
Figure 8.17: Von Mises stresses of the optimization result with inclined ribs.



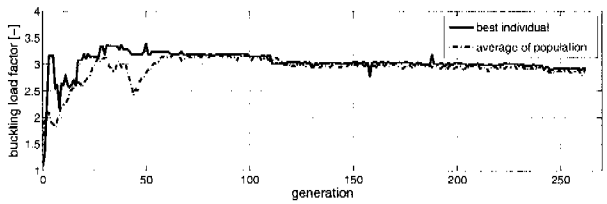
(a) fitness



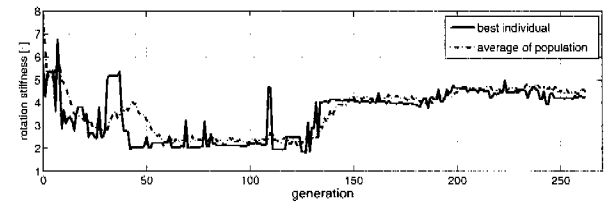
(b) mass



(c) stress



(d) stability



(e) stiffness

Figure 8.18: Convergence plots of the spar optimization with inclined ribs.

8.8 Conclusion

A graph-based representation concept is developed which is based on the specification tree of the CAD-software CATIA V5. Consequently, not only the shape, which can also be optimized by built-in optimization algorithms, but also the topology of mechanical structures can be optimized. The concept of user-defined features is perfectly suited to provide a 'library' of design entities which can be arranged in variable number within the design domain.

Due to its generic approach this EC framework is applicable to a wide variety of optimization problems. If only the shape of the structure should be optimized and the topology remains unchanged, i.e. no user-defined features are used, the setup of an optimization is even simpler. Once a mechanical structure is modeled only the naming rules need to be applied. After running the initializer program and adjusting the gene limits the optimization process can be immediately started.

The bridge-like minimum compliance problem is optimized and compared to a similar EA-based optimization performed by König [80]. The optimization result obtained by the new graph-based representation concept is superior for the optimization problem at hand. Furthermore, the optimization of a wing spar demonstrates a possible field of application for the presented concept. The representation concept is not limited to a maximum number of different design entities so that arbitrary mechanical structures can be optimized. Only some modeling rules have to be considered, but none of these rules generally limits the modeling possibilities.

The optimization of the spar structure is an example for a complex optimization task. The ribs and the cutouts are modeled in a flexible way since they depend on the adjacent geometry of the structure. Thus, complex mechanical structures are accessible to numerical optimization by the proposed CAD-representation concept which hardly could be optimized by other methods. The concept can be further developed in several ways.

- The applied variation operators are general in their nature, i.e., they are applicable to every mechanical structure. The optimization performance could be improved if variation operators implicitly including domain knowledge would be applied. As an example, a crossover operator could be developed which is based on a plane in the three-dimensional space and exchanges all design entities which are located on the same side of this plane from two arbitrary parent solutions.

- The variation operator rates are constant during the entire optimization process what might adversely affect the optimization performance. In the beginning of the optimization process variation operators changing the topology of the mechanical structure should be applied quite often, whereas their variation operator rates should be reduced later

in the optimization process. Topology variation operators can hardly find better solutions if the optimization process stagnates close to a local optimum. Instead, variation operators contributing to the fine tuning of the optimization variables should be forced in order to converge closer to the local optimum.

- The implementation is on a relatively low level in terms of usability. A graphical user interface (GUI) could be implemented in order to simplify the application of the EC framework.

- The presented graph-based representation concept is designed to meet the requirements of structural optimization of heavily constrained mechanical structures. Already minor changes of parameter values can lead to infeasible or even illegal structures whose specification trees cannot be updated anymore without producing topological errors. It would be interesting to enhance the concept to a freer topology optimization framework, i.e., the variation operators could also exchange entire subtrees of the original specification trees which contain different substructures themselves (similar to GP approaches). However, the formulation of an optimization problem with so few restrictions might be very challenging and it is likely that only simple mechanical features could be used for such an optimization.

Chapter 9

Conclusions and Outlook

9.1 Conclusions

This thesis investigates several aspects of numerical optimization in the field of structural engineering. The development of CFRP motorcycle racing rims is presented. The stacking sequence optimizations of the rims are an important step in the entire product development process and finally contribute to high-performance rear and front rims which proved their quality in race track tests.

The identified deficiencies are addressed and improved with the development of an adaptive optimization algorithm and the introduction of a variable-length representation particularly suited for global laminate optimization. AORCEA - the Adaptive Operator-Rate Controlled Evolutionary Algorithm - is tested on several unconstrained and constrained well-known benchmark functions and also the optimization of a steel trellis motorbike frame is treated. The adaptive algorithm proves to be efficient compared to a non-adaptive reference algorithm, i.e., the application of adaptive optimization strategies can decisively contribute to an improved convergence behavior.

The variable-length representation for global composite laminate optimization combining EAs with a patch concept allows for addressing laminate optimization tasks which are difficult to be tackled with traditional methods. Since the laminate layers are treated as basic design entities, they can be arranged independently in the design space. Thus, the stacking sequence can be perfectly adapted to the loading of the laminate structure. For a benchmark eigenfrequency optimization the variable-length representation is superior compared to the traditional representation approach which is based on boolean variables switching single layers on or off. These results indicate that the variable-length representation leads to an improved optimization performance although a generalization to other optimization problems is not necessarily permissible.

A straightforward application for graph-based representations is the optimization of truss structures due to the obvious similarity of mathematical graphs and truss structures. The concurrent optimization of topology, geometry, and sizing is rendered possible and it is completely independent from any kind of ground structure. Only the loaded and clamped nodes must be given and the optimization method itself tries to fit an optimum topology with optimum geometry and sizing into the given design space. The investigated optimization examples demonstrate the quality of the method leading to innovative topologies which hardly could be found with state-of-the-art optimization methods.

Also a novel graph-based representation concept for the optimization of laminated composites structures is introduced. It is able to concurrently optimize size, shape, position, number, orientation, and any other material related property of fiber-reinforced patches. The definition of zones allows for an almost arbitrary inclusion of domain knowledge and the complexity of the optimization problem can be influenced independent from the granularity of the underlying FE-mesh. The presented applications demonstrate the methods ability to find good quality results in constrained optimization problems.

Finally, a graph-based representation concept is developed which is based on the specification tree of the CAD-software CATIA V5. The shape but also the topology of arbitrary mechanical structures can be optimized. This is an advantage compared to the built-in optimization algorithms which are only able to perform pure shape optimization. The representation concept is not limited to a maximum number of different design entities, only some modeling rules have to be considered, but none of these rules generally limits the modeling possibilities. As an example, the optimization of a wing spar is demonstrated as a possible field of application.

When looking at the three graph-based representation concepts presented within this thesis, a kind of evolution of the mapping process can be observed. The mapping stage of the graph-based truss optimization (cf. Chapter 6) is rather simple since the genotype, i.e. the mathematical graph, and the corresponding phenotype, i.e. the corresponding truss structure, are essentially equal. In the case of the graph-based global laminate optimization the representation parameterizes the properties of the patches, but it also represents a kind of building rule determining the stacking sequence of the patches. Finally, the graph-based CAD representation is actually nothing else than a representation of a mapping procedure itself. The specification tree consists of mechanical features which can be understood as building rules of the structural part. The update mechanism sequentially processes these mechanical features and finally leads to the mechanical structure. This procedure can be considered as an example for 'evolution of evolution' which is a promising approach when searching for creative design solutions.

9.2 Outlook

The presented graph-based representation concepts are all tailored to typical structural optimization problems. However, similar representation concepts are employed in many other disciplines, e.g. electronic circuit design, scheduling, routing tasks, control systems, and process simulation. Basically any field of application where parts, rules, or events have to be arranged in a sequential and/or parallel manner can be addressed with EAs and suitable graph-based representation concepts. Consequently, the ideas presented in this thesis could definitely inspire similar fields of research in terms of graph-based representations and variation operator design. As an example, multi-body motion simulation[1] could be tackled with graph-based representations in order to optimally combine several components, e.g. pneumatics, hydraulics, electronics, to mechanical assemblies or mechanisms showing above-average performance in terms of natural frequencies and mode shapes.

The thesis is concluded by outlining some ideas for further research in the field of structural optimization.

- **Development of adaptive EAs for graph-based representation concepts.** Since the main focus of this thesis is laid on novel representation concepts, the development of (self-) adaptive EAs is neglected. Only the development of AORCEA contributes to an improved optimization performance. Further investigations in the field of adaptive optimization algorithms in particular for graph-based representation concepts which typically apply topology, shape, and sometimes also sizing variation operators could contribute to an improved optimization performance.

- **Hybrid optimization algorithms.** A promising approach to further improve the solution quality could be to develop hybrid optimization algorithms which combine the qualities of the stochastic EAs and the gradient-based search methods. For optimization problems with continuous optimization variables it is hardly possible to exactly locate an at least local optimum. Either manually adjusted or self-adaptive mutation step sizes could improve the local search, but EAs can hardly compete with gradient-based search methods in these search states. Alternatively, some approaches based on fitness landscape analysis as e.g. presented in Puisa [105] could be investigated. The optimization performance could be further improved by using databases or lookup tables storing already evaluated solutions which can be used as a basis for some fitness estimation models.

[1] Software package ADAMS, http://www.mscsoftware.com

- **Mesh-based topology optimization.** After the concept of graph-based representations for truss structures, global laminate optimization, and specification tree optimization of CAD-modeled mechanical structures one may think of further applications of graph theory in computational structural optimization. The laminate optimization bears a resemblance to mesh-based topology optimization where graph theory could give novel inputs as well. Such novel approaches could be component-based (e.g. panels, beams, ribs, webs, or stringers) in order to include as much design knowledge as possible into the optimization process.

- **CAD topology optimization workbench.** The CAD-software CATIA V5 is organized in a modular way, whereas for different disciplines appropriate workbenches are available. The graph-based specification tree representation could be integrated in a stand-alone workbench providing the necessary user interfaces to simply define topology optimization problems.

- **CAD boundary representation.** In contrast to the 'evolution of evolution' principle which is heavily knowledge-based, the boundary representation, i.e. the resulting shape after processing the specification tree, could be parameterized. As basic design entities vertices, edges, wires, faces, solids, etc. and their underlying geometric entities (points, curves, and surfaces) could be parameterized. This approach to CAD-based design optimization is addressed in a further research project at the Centre of Structure Technologies.

Appendix A

UniGene - the universal genotype

The motivation to develop the universal genotype arose from the need for a heterogeneous representation of mechanical structures which are most often built in CAD or FEM systems. The defining parameters are typically of different types such as real, integer, or boolean, and they need to have lower and/or upper limits in order to implicitly include constraints into the representation.

The universal genotype consists of a collection of different gene types which is tailored to the optimization problem at hand. Typically, the collection is a heterogeneous list of genes of the following types:

- *Float-gene* represents a real-valued design parameter with optional and arbitrary lower and upper limits.
- *Integer-gene* represents an integer-valued design parameter with optional and arbitrary lower and upper limits.
- *Bool-gene* represents a binary design parameter with state *true* or *false*.
- *Float-list-gene* represents a real-valued design parameter with arbitrary alleles.
- *Const-float-list-gene* represents a real-valued design parameter with equidistant alleles between a lower and an upper limit.
- *String-list-gene* represents a discrete design parameter of arbitrary values upon which no order or norm can be applied.

Beneath the actual value of a gene some further information needs to be provided in order to properly define a gene. At first, each gene has a parameter defining whether it is active or not. A gene can be set inactive, if

it should no longer be subject to changes. Furthermore, for some gene types it needs to be defined whether a lower and an upper limit value exist and, if so, the limit values itself.

For each gene type uniform and Gaussian mutation are implemented, thus, needing some defining parameters. The ϵ-value is required for uniform mutation, where a new value is assigned to the gene which is randomly chosen within the given limits of the gene. In case the lower and/or the upper limit is missing, the ϵ-value is added and/or subtracted to the current gene value v in order to define the uniform mutation interval $[v-\epsilon, v+\epsilon]$.

The purpose of Gaussian mutation is to slightly change the current gene value v and seek for improved solutions close to the parent solution. The Gaussian mutation can be easily defined as

$$v' = v + \mathcal{N}(0, \sigma), \tag{A.1}$$

where σ defines the standard deviation of the zero-mean normal distribution. For bounded genes the new value v' could fall out of the limits. In this case two different strategies are applied; either the new value is moved exactly to the limit, or it is mirrored back until it falls into the given limits. For genes where no ordering can be defined, i.e. the bool- and the string-gene, Gaussian mutation cannot be applied straight forward. Small variations are therefore introduced according to the following rules for bool-genes

$$v'_{bool} = \begin{cases} \neg v_{bool} & : \quad |\mathcal{N}(0,\sigma)| > 1 \\ v_{bool} & : \quad else \end{cases} \tag{A.2}$$

and for string-genes

$$v'_{string} = \begin{cases} UniformMutation\,(v_{string}) & : \quad |\mathcal{N}(0,\sigma)| > 1 \\ v_{string} & : \quad else \end{cases} \tag{A.3}$$

Finally, float-, integer-, float-list-, and const-float-list-genes can further be provided with so-called cyclic properties. This is suitable if between two possible gene values a distance but no absolute order can be defined, as e.g. for angle values.

An example definition for each gene type is given below. The description corresponds to the definition as it is used in the C++ optimization environment (cf. Section 2.6).

- *[float_gene true 1.2 true 0.0 true 2.5 1.0 0.25 false]*
  denotes a float-gene which is active and has a default value of 1.2. The second and the third *true* values indicate that a lower as well as an upper limit exist with the corresponding values of 0.0 and 2.5. Further the mutation parameters $\epsilon = 1.0$ and $\sigma = 0.25$ are specified, whereas ϵ is useless in this case (both limits are active) and σ defines the standard deviation to be used for Gaussian mutation. The last boolean value indicates that the gene does not have cyclic properties.

- *[int_gene true 3 true 1 false 0 10 4 false]*
  denotes an int-gene which is active and has a default value of 3, an active lower limit of 1, and no upper limit. The value $\epsilon = 10$ is required to define the uniform mutation range and $\sigma = 4$ defines the standard deviation of the Gaussian mutation. Finally, this gene does not have cyclic properties.

- *[bool_gene true true 2.0]*
  denotes a bool-gene which is active and has default value *true*. Here, the standard deviation $\sigma = 2.0$ defines a kind of probability for changing the current gene value.

- *[float_list_gene true 4 3 0.0 1.0 false 1.2 3.2 6.9 9.8]*
  denotes a float-list-gene which is active and has four alleles, where the default value of the gene is the one with index 3, i.e. 9.8 (indexing starts from 0). The mutation parameters $\epsilon = 0.0$ (useless for this gene) and $\sigma = 1.0$ as well as the definition of cyclic properties follow. Finally, the four possible allele values are given in ascending order.

- *[const_float_list_gene true 45.0 true 0.0 true 360.0 24 0.0 15.0 true]*
  denotes a const-float-list-gene which is active and has a default value of 45.0. The gene has an active lower as well as an active upper limit and the number of intervals is set to 24 leading to allele values [0.0, 15.0, ... , 345.0, 360.0]. Since the gene is fully bounded the ϵ-value is not used, but the σ-value defines the standard deviation of Gaussian mutation. The cyclic property is *true* as it is for example useful for the representation of discrete angle values.

- *[string_gene true 3 1 1.0 blue green red]*
  denotes a string-gene which is active with three allele values, whereas the default value has index 1, i.e. *green* (indexing starts from 0). The standard deviation σ is stated and the list of alleles is given in the end.

In the original implementation of König [80] and Wintermantel [89] the universal genotype concept is extended with a set of crossover and mutation operators acting on a vector-like genotype consisting of the above-mentioned gene types. However, the basic gene types presented here are also suited to be components of any other representation structure, such as variable-length vectors, trees, or arrays. The implementation of the appropriate reproduction operators is therefore always problem- and representation-dependent.

Appendix B

FELyX - The Finite Element LibrarY eXperiment

FELyX is an object oriented finite element library written in C++. It is primarily designed to meet the demands of researchers and developers who want to have a flexible and easy-to-handle finite element library allowing them to implement their own ideas and projects quickly. FELyX provides all features for structural static and eigenfrequency analysis of complex real-world structures, namely the relevant elements, an interface to a commercial FEA package, smart node reordering strategies, and efficient solver algorithms. Beyond this, FELyX is an open source project [150] which is consequently further developed in order to enhance the library's simulation capabilities. The author is developer of FELyX and has provided several contributions to the project. The following explanations are based on the dissertation of König [80] – one of the main developers of FELyX.

B.1 Introduction to the Finite Element Method

The Finite Element Method (FEM) is a technique for the numerical solution of field problems. The solution to a field problem, which is mathematically described by differential equations or integral expressions, is the spatial distribution of one or more dependent variables. Finding a solution for these equations for complex spatial domains can be demanding, but FEM achieves the solution of field problems in any arbitrary spatial domain.

The idea of the FEM is to subdivide a given complex spatial domain into smaller elements, whose behavior is well understood. The variation of the field quantity in each element is assumed to be small and is therefore mostly described by polynomials up to order two or three. Nevertheless, the actual variation within the region spanned by a single element is generally more complicated, hence, FEM normally provides only approximate solutions.

Within a single element the field quantities are given as functions of

unknown values at the nodes, whereas the nodes are well-defined points mostly lying on the border of the element domain. In general, numerous elements are connected at their nodes to form the mesh which represents a surrogate model of the real domain and its physical behavior. Such a model is in a mathematical sense a system of linear equations to be solved for the unknown nodal values [157].

Although FELyX provides a framework for general finite element analyses, the main interest lies in the field of static structural analysis. Thus, the following explanations will exclusively focus on static structural analysis for brevity reasons.

The stress state in a loaded structure is a typical field problem. The preprocessing steps of a structural finite element analysis are as follows. First, the FE-mesh must be built in order to discretize the structure in finite elements. Afterwards, all nodal coordinates are determined and all elements are assigned their nodes, hence, all elements are geometrically well-defined. Furthermore, all elements must contain the material properties of the domain they span. After assigning the geometrical and physical properties the surrogate model of the structure is complete.

The loads applied on the real structure must be translated to the surrogate model. Distributed loads like pressure must be mapped to forces acting on nodes and geometric boundary conditions are defined as predefined displacements of single nodes. One can distinguish two different types of such boundary conditions. A geometric boundary condition is called homogeneous if it specifies a zero displacement, else it is called inhomogeneous.

After completing the surrogate model of the loaded structure the appropriate mathematical functions can be established. Each finite element must fulfill the equations of equilibrium and all stresses acting on the boundaries of an element are represented as statically equivalent nodal forces. The stress state within the element is represented by a function, mapping nodal displacements to nodal forces. This mapping is often given as a system of linear equations, expressed by the so-called Element Stiffness Matrix (ESM)

.

The present load case is given as the following matrix equation containing the displacement vector $\{u\}$, the force vector $\{f\}$, and the Global Stiffness Matrix (GSM) $[K]$.

$$[K]\{u\} = \{f\} \tag{B.1}$$

The GSM $[K]$ is an assembly of the individual ESMs. The nodal displacement vector $\{u\}$ is unknown except for the specified geometric boundary conditions. If homogeneous boundary conditions $u_i = 0$ appear, the linear equations they appear in are trivial and can be removed from the global system of linear equations. Inhomogeneous boundary conditions $u_i \neq 0$ can be represented as a superposed load case $\{f_d\}$ and must be added to $\{f\}$.

The force vector $\{f\}$ contains the nodal forces derived from the applied load case and is therefore known.

Finally, Equation B.1 can be solved for the vector of nodal unknowns $\{u\}$. Then the stress state within the individual elements is a function of the nodal displacements $\{u\}$ and is easily evaluated once the primary solution is known. For layered shell elements also further postprocessing analysis is possible by evaluating failure criteria, namely Maximum Stress [152], Tsai-Hill [153], Tsai-Wu [98], and Hashin [154].

B.2 Structure of FELyX

FELyX is implemented in the programming language C++ which is a widely available and standardized language providing high levels of abstraction for object-oriented and generic programming. Furthermore, there is a large amount of libraries of generic data containers and algorithms as for example the Standard Template Library (STL) [158] or the Boost Libraries [139]. With all these features, C++ supports efficient programming techniques.

A finite element library must be able to store large amounts of data, to hold the entire information of a discretized surrogate model. FELyX stores all this data in STL vectors templated with the appropriate data objects. The various objects of a FELyX surrogate model are the nodal coordinates, the elements, the material properties, the loads, the boundary conditions, and the nodal coordinate systems. The dependencies of these objects are shown in Figure B.1. The different lists in Figure B.1 symbolize the STL vectors containing objects of the FELyX library. The arrows stand for the data access of individual objects. This access is implemented with pointers what is substantially faster than indices-lookup mechanisms and in such an architecture data is not stored redundantly.

A central object in the architecture of FELyX is the class `Element`. Every element has a unique set of data defining its individual physical behavior. FELyX organizes this data in three subsets, namely the material, the general properties, and the nodes. As can be seen in Figure B.1 different elements can access the same subsets of data. An object `Material` contains the physical material properties and methods providing different stress-strain relations. The class `PropertySet` is a pool for element-type dependent data such as thickness of a shell, cross-sectional area of a beam, etc., and the shape of an element is defined by the coordinates of its nodes.

Besides its coordinates a node can have boundary conditions applied on it. Typically, these boundary conditions refer to the global cartesian coordinate system. If the boundary conditions should be applied in other directions, the node can be assigned with an additional coordinate system.

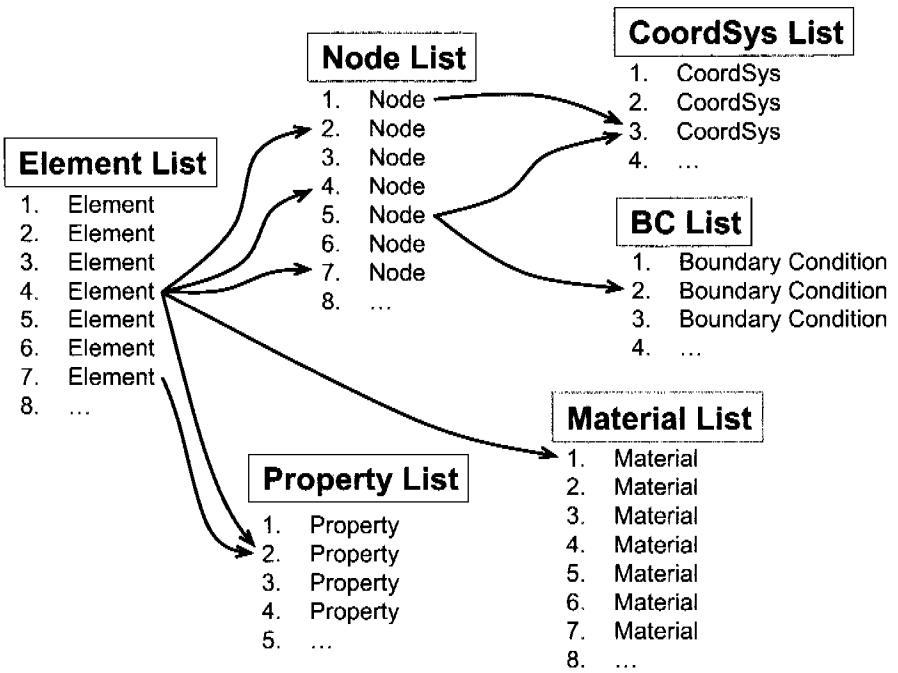


Figure B.1: Data structure of FELyX. The arrows represent direct access to objects.

B.3 Features

B.3.1 Element implementations

In FELyX the common elements for static structural analysis are implemented. In two dimensional space:

- 2-node link element with two translational degrees of freedom (DOF) per node
- 2-node beam element with two translational and one rotational DOFs per node
- Triangular 6-node plane element with quadratic displacement behavior and two translational DOFs per node
- Quadrilateral 4-node plane element with linear displacement behavior and two translational DOFs per node
- Quadrilateral 8-node plane element with quadratic displacement behavior and two translational DOFs per node

In three dimensional space:

- 2-node link element with three translational DOFs per node
- 2-node beam element with three translational and three rotational DOFs per node
- Tetrahedral 10-node element with quadratic displacement behavior and three translational DOFs per node
- Quadrilateral 8-node element with linear displacement behavior and three translational DOFs per node
- Quadrilateral 20-node element with quadratic displacement behavior and three translational DOFs per node
- Triangular 6-node and quadriliteral 8-node shell element with quadratic displacement behavior, three translational DOFs and three rotational DOFs per node
- Triangular 6-node and quadriliteral 8-node layered shell element with quadratic displacement behavior, three translational DOFs and three rotational DOFs per node.

All elements are derived from an abstract base class `Element`. This class contains all data members and functions which are common to most of the derived elements. The derived elements only contain data members and functions that are unique to their special type, e.g. the number of nodes.

B.3.2 Linear solvers

FELyX provides a direct (*Gauss/Cholesky–type*) and an iterative solver for the evaluation of the linear equation system (B.1). Additionally, the PARDISO solver (Parallel Sparse Direct Linear Solver) is integrated which was developed at the University of Basel [159] and is included in Intel's MKL library.

FELyX performs also modal analysis of structures. For the determination of eigenfrequencies the Lanczos [160] algorithm is implemented.

B.3.3 Interface to ANSYS

One of the major requirements for FELyX is its ability to handle complex real-world structures. To achieve this goal, an interface to the commercial FEM program ANSYS is implemented. Starting from a CAD design of a structure, the geometry is imported to ANSYS, where all the preprocessing of the FEM model takes place. This includes the discretization of the geometry, definition of material and element properties as well as the application of boundary conditions. The resulting FE model is then exported

to a text file, using the ANSYS archive format. This file can be read into FELyX, storing elements, nodes, materials, element properties, coordinate systems, and boundary conditions in the appropriate STL vectors as introduced in Section B.2. All ANSYS interface classes are derived from general IO classes, facilitating the implementation of interfaces to other programs.

Appendix C

Mathematical graph theory and the Boost Graph Library (BGL)

C.1 Basic notions of graph theory

It would be immoderate to give a thorough introduction to mathematical graph theory at this point, since it is a field of research of inconceivable complexity. Thus, this appendix only covers the most important notions and concepts which are used in the scope of this thesis.

In literature graphs are defined in slightly different ways, e.g. Foulds [161], Chen [162], or Balakrishnan and Ranganathan [136]. A brief definition is given here.

Def. 11 *A graph G(V,E) is an ordered pair, where V is a finite, nonempty set whose elements are termed vertices, and where E is a set of unordered pairs of distinct vertices of V. Each element e=(u,v) ∈ E (where u, v ∈ V), is called an edge and is said to be incident with its vertices u and v. Also, the vertices u and v are then incident with e.*

There are some other important notions which will be explained by an example. Consider the graph *G(V,E)* in Figure C.1 in which

$$\begin{aligned} V &= \{v1, v2, v3, v4, v5\} \\ E &= \{(v1, v2), (v1, v3), (v2, v2), (v2, v3), \\ &\quad (v3, v4)_1, (v3, v4)_2\} \end{aligned} \tag{C.1}$$

Basically, vertices u and v in V are *adjacent* to each other if, and only if, there is an edge in E with u and v as its endpoints. E.g., the vertices $v1$ and $v2$ are adjacent, since they are connected by the edge $e1$. A vertex having no incident edges, i.e. it is not connected to any other vertex in V,

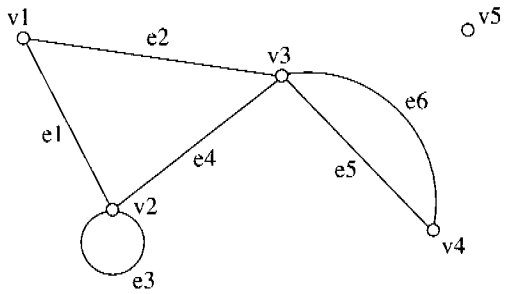


Figure C.1: Graph *G(V,E)*.

is called an *isolated vertex* (e.g. $v5$). Two edges of a graph are termed a set of *parallel edges*, if they have the same endpoints, e.g. $e5 = (v3, v4)_1$ and $e6 = (v3, v4)_2$. An edge for which the two endpoints are the same, e.g. $e3 = (v2, v2)$, is called a *loop* at the common vertex. In *undirected* graphs edges are unordered pairs, therefore (u, v) and (v, u) represent the same edge, whereas *directed* and *bidirectional* graphs distinguish between *source* and *target* vertex. Finally, the number of vertices and edges in the graph *G(V,E)* are denoted the *order* and the *size*, respectively. The graph *G(V,E)* in Figure C.1 has order $n(G(V, E)) = 5$ and size $m(G(V, E)) = 6$.

Moreover, it is possible to assign properties to graphs, i.e. to its vertices and edges. A graph is said to be *labeled*, if its $n(G)$ vertices are distinguished from one another by labels $v1, v2, \ldots, vn$ as it is done in the example. Furthermore, arbitrary additional properties can be attached to each vertex as it is for example done for the so-called graph coloring problem, where each vertex is colored. Analogously, the concept of a *weighted graph* associates a nonnegative number $w(e)$ to each edge which is called *edge weight*. The sum of all edge weights is therefore the total weight of the graph.

C.2 Adjacency matrix

The adjacency matrix is a storage format of mathematical graphs. Additionally, it serves the purpose of visualizing the structure of a graph, i.e., the adjacency matrix shows which vertices are connected by edges. As an example, the effect of genetic topology and sizing operators can be shown by means of the adjacency matrix.

In the book of Foulds [161] the traditional adjacency matrix is defined in the following way.

Def. 12 *The adjacency matrix* $A = (a_{ij})_{n \times n}$, *of a vertex-labeled graph G, with n(G) vertices, is the matrix in which* $a_{ij} = 1$ *if vertex* v_i *is adjacent to*

v_j in G and $a_{ij} = 0$ otherwise, where v_i and v_j are vertices of G.

In some cases it is reasonable to replace $a_{ij} = 1$ by the edge weights $a_{ij} = w_{ij}$ of the respective edge in order to insert more information into the adjacency matrix of the graph. E.g., the traditional and the weighted adjacency matrices of the simple graph shown in Figure 6.1 are

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}, \qquad A_w = \begin{bmatrix} 0 & w_{12} & 0 & w_{14} \\ w_{21} & 0 & w_{23} & w_{24} \\ 0 & w_{32} & 0 & w_{34} \\ w_{41} & w_{42} & w_{43} & 0 \end{bmatrix}. \tag{C.2}$$

It is quite obvious that A and A_w, respectively, must be symmetric, i.e. $w_{ij} = w_{ji}$. Furthermore, the entries on the leading diagonal equal zero due to the simplicity of the graphs not allowing loops. A lot more information is contained in this kind of graph description, e.g. a block diagonal matrix indicates a non-connected graph (subgraphs exist), but in the scope of this thesis the given definition is sufficient.

C.3 Cuthill-McKee Reordering

The basic purpose of the Cuthill-McKee reordering algorithm [163] is to reduce the bandwidth of a sparse symmetric matrix, e.g., the adjacency matrix. The bandwidth of an undirected graph and its adjacency matrix, respectively, is the maximum labeling distance between two adjacent vertices. In case of a graph $G(V, E)$ with labeled vertices from 1 to $n(G)$ the bandwidth $B(G)$ is defined as

$$B(G) = max\{\mid label[u] - label[v] \mid \quad \mid u, v \in V\}. \tag{C.3}$$

The bandwidth of the adjacency matrix can thus be minimized by reordering the labels assigned to each vertex, whereas a starting vertex needs to be provided which generally is labeled 0.

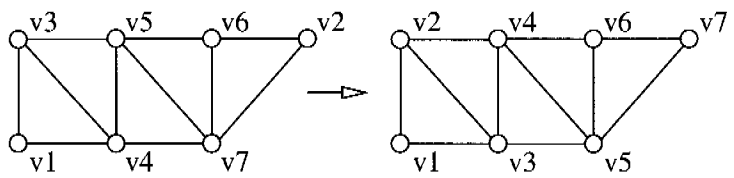


Figure C.2: Truss topologies with arbitrary (left) and reordered (right) vertex labeling (starting vertex $v1$).

In Figure C.2 a sample truss topology is presented with arbitrary vertex labeling on the left and reordered labeling on the right. The corresponding adjacency matrices are given below. The zeros beneath the leading diagonal are omitted for better perceptibility.

$$A_{arb} = \begin{bmatrix} 0 & & 1 & 1 & & & \\ & 0 & & & & 1 & 1 \\ 1 & & 0 & 1 & 1 & & \\ 1 & & 1 & 0 & 1 & & 1 \\ & & 1 & 1 & 0 & 1 & 1 \\ & 1 & & & 1 & 0 & 1 \\ & 1 & & 1 & 1 & 1 & 0 \end{bmatrix} \rightarrow A_{reordered} = \begin{bmatrix} 0 & 1 & 1 & & & & \\ 1 & 0 & 1 & 1 & & & \\ 1 & 1 & 0 & 1 & 1 & & \\ & 1 & 1 & 0 & 1 & 1 & \\ & & 1 & 1 & 0 & 1 & 1 \\ & & & 1 & 1 & 0 & 1 \\ & & & & 1 & 1 & 0 \end{bmatrix}$$

The bandwidth of the first adjacency matrix is $B(A_{arb}) = 5$ and the bandwidth of the reordered adjacency matrix is $B(A_{reordered}) = 2$. Besides the original Cuthill-KcKee reordering also a reversed implementation exist which generally outperforms the original method in terms of the achieved bandwidth reduction. Thus, within this thesis the reverse Cuthill-McKee reordering strategy is applied.

C.4 BGL - the Boost Graph Library

For the implementation of mathematical graph structures in the object-oriented programming language C++ the Boost Graph Library [139] (BGL) is perfectly suited since it provides the required data structures and graph processing algorithms.

The efficient determination of neighboring vertices is a particularly useful feature. Instead of checking the entire data set to determine a neighboring vertex, the adjacent vertices can directly be accessed through the connecting edges. The following explanations are based on Siek et al. [137].

C.4.1 Graph classes and adaptors

The BGL contains two primary graph classes, namely the *adjacency_list* and the *adjacency_matrix*. Several template parameters are required: *EdgeList*, *VertexList*, *Directed*, *VertexProperties*, *EdgeProperties*, and *GraphProperties*.

- *EdgeList* and *VertexList* specify the classes used to store the vertex list and edge list for the graph. These parameters allow tradeoffs between traversal speed and insertion/removal speed and tradeoffs in memory consumption. In addition, the *EdgeList* parameter determines whether parallel edges may be inserted into the graph.
- *Directed* specifies whether the graph is directed, undirected, or bidirectional.

- *VertexProperties*, *EdgeProperties*, and *GraphProperties* specify the property types that are attached to the vertices, to the edges, and to the graph itself.

Furthermore, the BGL also includes a large number of graph adaptors which create a modified view of a graph. The *reverse_graph* adaptor reverses the edge directions of a directed graph. The *filtered_graph* is an adaptor that creates a view of a graph where two predicate function objects control whether vertices and edges from the original graph appear in the adapted graph, or whether they are hidden.

C.4.2 Property maps

Graphs become useful as models for particular problem domains by associating objects and quantities to vertices and edges. In the BGL these objects or quantities are referred to as *properties*. A wide variety of such properties can be associated, e.g. as data members of a struct or separate arrays indexed by vertex or edge number.

The access to these properties is defined by the so-called *property map* concept. A property map is an object that provides a mapping from a set of key objects to a set of value objects. The property map concepts specify only three functions in order to have a uniform syntax for accessing different kinds of properties. The *get(p_map, key)* functions returns the value object for the *key*, *put(p_map, key, value)* assigns the *value* to the value object associated with the *key*, and *p_map[key]* returns a reference to the value object.

C.4.3 Graph traversal

BGL implements various iterator types similar to the Standard Template Library (STL) for efficient data structure processing. In general, the graph abstraction consists of several different kinds of collections: the vertices and edges for the graph and the out-edges, in-edges, and adjacent vertices for each vertex. There are five kinds of graph iterators:

- A *vertex iterator* is used to traverse all the vertices of the graph.
- An *edge iterator* is used to traverse all edges of the graph.
- An *out-edge iterator* is used to access all of the out-edges for a given vertex.
- An *in-edge iterator* is used to access the in-edges for a given vertex.
- An *adjacency iterator* is used to provide access to the vertices adjacent to a given vertex.

The BGL is a very powerful library providing a lot more graph algorithms. Further information can be found in Siek et al. [137].

List of Tables

List of Figures

Bibliography

[1] W. Spendley, G. R. Hext, and F. R. Himsworth. Sequential application of simplex designs in optimization and evolutionary operation. *Technometrics*, 4:441–461, 1962.

[2] M. J. D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *Computer J.*, 7:155 162, 1964.

[3] R. H. Myers and D. C. Montgomery. *Response Surface Methodology, Process and Product Optimization Using Designed Experiments.* John Wiley & Sons, Inc. New York, 1989.

[4] A.-L. Cauchy. Methode générale pour la résolution des systemes d'éqations simultanées. *Compt. Rend. Acad. Sci.*, 25:536–538, 1847.

[5] R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *Comp, J.*, 7:149–154, 1964.

[6] J. Nocedal and S. J. Wright. *Numerical optimization.* Springer, New York, 1999.

[7] M.P. Bendsøe. *Optimization of structural topology, shape, and material.* Springer-Verlag, Berlin, Heidelberg, New York, 1995.

[8] M.P. Bendsøe and N. Kikuchi. Generating optimal topologies in structural design using a homogenization method. *Computer Methods in Applied Mechanics and Engineering*, 71:197–224, 1988.

[9] M.P. Bendsøe and O. Sigmund. *Topology Optimization: Theory, Methods and Applications.* Springer-Verlag, Berlin Heidelberg, 2003.

[10] M. Kocvara. Modern optimization algorithms in topology design. In *Computer Aided Topology Optimization Seminar, Bayreuth, Germany*, 2000.

[11] F. Vogel. Implementation of the Ansys topology optimizer. In *Computer Aided Topology Optimization Seminar, Bayreuth, Germany*, 2000.

[12] K. Maute and E. Ramm. Adaptive topology optimization. *Structural Optimization*, 10:100–112, 1995.

[13] G.I.N. Rozvany. *Structural design via optimality criteria: the Prager approach to structural optimization.* Kluwer Academic Publishers, Dordrecht, 1989.

[14] C. Mattheck. *Design in der Natur, der Baum als Lehrmeister.* Rombach GmbH + Co Verhlagshaus KG, 1992.

[15] R. Kicinger, T. Arciszewski, and K. De Jong. Evolutionary computation and structural design: A survey of the state-of-the-art. *Computers & Structures*, 83(23-24):1943–1978, 2005.

[16] Th. Bäck, H.-P. Schwefel, and U. Hammel. Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1(1):3–17, 1997.

[17] M. Schoenauer and Z. Michalewicz. Evolutionary computation: an introduction. *Control and Cybernetics*, 26(3):307–338, 1997.

[18] J.H. Holland. Genetic algorithms and the optimal allocation of trials. *SIAM Journal on Computing*, 2(2):88–105, 1973.

[19] J.H. Holland. *Adaptation in natural and artificial systems.* University Michigen Press, Ann Arbor, MI, 1975.

[20] D.E. Goldberg. *Genetic algorithms in search, optimization and machine learning.* Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1989.

[21] K. De Jong. *The Analysis of the Behaviour of a Class of Genetic Adaptive Systems.* PhD thesis, Department of Computer Science, University of Michigan, Ann Arbor, Michigan, 1975.

[22] L. Davis, editor. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, 1991.

[23] J.J. Grefenstette. Optimization of control parameters for genetic algorithms. *Transactions on Systemes, Man and Cybernetics*, 16(1):122–128, 1986.

[24] L.J. Fogel. *Biotechnology: Concepts and applications.* Prentice-Hall, Englewood Cliffs, NJ, 1963.

[25] L.J. Fogel, A.J. Owens, and M.J. Walsh. *Artificial intelligence through simulated evolution.* John Wiley, New York, 1966.

[26] D. Fogel. *Evolving Artificial Intelligence.* PhD thesis, University of California, San Diego, California, 1992.

[27] G. H. Burgin. On playing two-person zero-sum games against non-minimax players. *IEEE Trans. Syst. Sci.. Cybern.*, 5(4):369–70, Oct. 1969.

[28] J. W. Atmar. *Speculation on the evolution of intelligence and its possible realization in machine form.* PhD thesis, New Mexiko State University, Las Cruses, 1976.

[29] I. Rechenberg. *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution.* Fromann-Holzboog, Stuttgart-Bad Cannstatt, 1973.

[30] H.-P. Schwefel. *Evolutionsstrategie und numerische Optimierung.* PhD thesis, TU Berlin, 1975.

[31] Th. Bäck. *Evolutionary algorithms in theory and practice: Evolution strategies, evolutionary programming, genetic algorithms.* Oxford Univerity Press, New York, 1996.

[32] H.P. Schwefel. *Numerical Optimization of Computer Models.* John Wiley & Sons, New York, 1981.

[33] Th. Bäck. *Evolutionary algorithms in theory and practice.* Oxford University Press, 1995.

[34] J. Koza. *Genetic programming: On the programming of computers by means of natural selection.* MIT Press, Cambridge, MA, 1992.

[35] G. L. Steele, S. E. Fahlman, and D. G. Bobrow. *Common LISP: the language.* Digital Press, Bedford, Massachusetts, 2 edition, 1990.

[36] J. M. De La Cruz, A. Herrán-González, J. L. Risco-Martín J, and B. Andrés-Toro. Hybrid heuristic and mathematical programming in oil pipelines networks: Use of immigrants. *J Zhejiang Univ SCI*, 6A(1):9 19, 2005.

[37] J.D. Schaffer. *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms.* PhD thesis, Vanderbilt University, 1984.

[38] J.D. Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In J. J. Grefenstette, editor, *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, pages 93–100, Pittsburgh, PA, 1985.

[39] Eckhart Zitzler. *Evolutionary algorithms for multiobjective optimization: methods and applications.* PhD thesis, Computer Engineering and Networks Laboratory, ETH Zurich, November 1999.

[40] J. Horn, N. Nafpliotis, and D.E. Goldberg. A niched pareto genetic algorithm for multiobjective optimization. In *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress Bibliography 115 on Computational Computation*, volume 1, pages 82–87, Piscataway, NJ, 1994.

[41] C.M. Fonseca and P.J. Fleming. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 416–423, San Mateo, California, 1993.

[42] P. Hajela. Stochastic search in structural optimization: Genetic algorithms and simulated annealing. In *Structural Optimization: Status and Promise*, volume 150, pages 611–637. Progress in Astronautics and Aeronautics, 1992.

[43] J. P. Cohoon, S. U. Hegde, W. N. Martin, and D. S. Richards. Punctuated equilibria: a parallel genetic algorithm. In J. J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms (ICGA'87)*, pages 148–54, Cambridge, MA, USA, 1987. Lawrence Erlbaum.

[44] M. A. Potter and K. A. De Jong. Cooperative coevolution: an architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1):1–29, 2000.

[45] P. J. Angeline and J. B. Pollack. Competitive environments evolve better solutions for complex tasks. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA'93)*, pages 264–70, San Matteo, CA, 1993. Kaufmann.

[46] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer, 1994.

[47] E. Sandgren, E. Jensen, and J. Welton. Topological design of structural components using genetic optimization methods. In *Proceedings of the Winter Annual Meeting of the American Society of Mechanical Engineers*, volume AMD-vol. 115, pages 31 43, Dallas, TX, 1990.

[48] E. D. Jensen. *Topological structural design using genetic algorithms.* PhD thesis, Purdue University, 1992.

[49] C.D. Chapman, K. Saitou, and M.J. Jakial. Genetic algorithms as an approach to configuration and topology design. *Journal of Mechanical Design*, 116(4):1005–1012, 1994.

[50] C.D. Chapman and M.J. Jakiela. Genetic algorithm-based structural topology design with compliance and topology simplification considerations. *Journal of Mechanical Design*, 118:89–98, March 1996.

[51] C. Kane, F. Jouve, and M. Schoenauer. Structural topology optimization in linear and nonlinear elasticity using genetic algorithms. In *21st ASME Design Automatic Conference*, Boston MA, 1995. ASME Press.

[52] Couro Kane and Marc Schoenauer. Topological optimum design using genetic algorithms. *Control and Cybernetics*, 25(5), 1996.

[53] H. Hamda, F. Jouve, E. Lutton, M. Schoenauer, and M. Sebag. Compact unstructured representations for evolutionary design. *Applied Intelligence*, 16(2):139 155, 2002.

[54] H. Hamda, O. Roudenko, and M. Schoenauer. Application of a multi-objective evolutionary algorithm to topological optimum design. In I. Parmee, editor, *Adaptive Computing in Design and Manufacture*. Springer Verlag, 2002.

[55] W. Dorn, R. Gomory, and H. Greenberg. Automatic design of optimal structures. *Journal Mechanique*, 3:25–52, 1964.

[56] P. Hajela and J. Lee. Genetic algorithms in truss topological optimization. *International Journal of Solids and Structures*, 32(22):3341 3357, 1995.

[57] S. D. Rajan. Sizing, shape, and topology design optimization of trusses using genetic algorithms. *J. Struct. Engrg.*, 121(10):1480–87, 1995.

[58] S. Rajeev and C. S. Krishnamoorthy. Genetic algorithms-based methodologies for design optimization of trusses. *J. Struct. Engrg.*, 123(3):350–8, 1997.

[59] I. A. Azid, A. S. K. Kwan, and K. N. Seetharamu. An evolutionary approach for layout optimization of three-dimensional truss. *Structural and Multidisciplinary Optimization*, 24:333–337, 2002.

[60] F. Erbatur, O. Hasançebi, I. Tütüncü, and H. Kılıç. Optimal design of planar and space structures with genetic algorithms. *Computers & Structures*, (75):209–24, 2000.

[61] W. Hansel and W. Becker. Layerwise adaptive topology optimization of laminate structures. *Engineering computations*, 16(7):841 851, 1999.

[62] J. Zowe, M. Kocvara, and M. P. Bendsøe. Free material optimization via mathematical programming. In T. M. Liebling and D. de Werra, editors, *Mathematical programming*, volume 79 of *B*, pages 445–466. Mathematical programming society, Elsevier Science B.V., 1997.

[63] R. Le Riche and R. T. Haftka. Improved genetic algorithm for minimum thickness composite laminate design. *Composites Engineering*, 5(2):143-61, 1995.

[64] G. A. E. Soremekun. Genetic Algorithms for composite laminate design and optimization. Master's thesis, Virginia Tech, February 1997.

[65] S. Nagendra, R. T. Haftka, and Z. Gürdal. Stacking sequence optimization of simply supported laminates with stability and strain constraints. *AIAA Journal*, 30(8):2132–2137, August 1992.

[66] N. Kogiso, L. T. Watson, Z. Gürdal, and R. T. Haftka. Genetics Algorithms with local improvement for composite laminte design. *Structural optimization*, 7(4):207–218, June 1994.

[67] S. Venkataraman and R. T. Haftka. Optimization of composite panels - a review. In *Proceedings of the 14th Annual Technical Conference of the American Society of Composites*, Dayton, OH, 1999.

[68] O. Pironneau. *Optimal shape design for elliptic systems.* Springer-Verlag, 1984.

[69] J. A. Bennet and M. E. Botkin, editors. *The optimum shape: automated structural design.* Plenum Press, New York, London, 1986.

[70] S.Y. Woon, O.M. Querin, and G.P. Steven. Structural application of a shape optimization method based on a genetic algorithm. *Structural and Multidisciplinary Optimization*, 22(1):57–64, 2001.

[71] W. M. Jenkins. Towards structural optimization via the genetic algorithm. *J Struct Eng*, 1991.

[72] J. Huang and G. M. Fadel. Heterogeneous flywheel modeling and optimization. *invited paper, Journal of Materials and Design*, 21:111–125, 2000.

[73] P. Pedersen. Optimal joint positions for space structures. *J Struct Eng*, 1987.

[74] D. E. Grierson and W. Pak. Optimal sizing, geometrical and topological design using a genetic algorithm. *Struct Optimiz*, 1993.

[75] J. S. Arora, editor. *Introduction to optimum design.* McGraw Hill, New York, 1989.

[76] D. E. Goldberg and M. Samtani. Engineering optimization via genetic algorithm. In *Proceedings of the Ninth Conference on Electronic Computation*, pages 471–82, University of Alabama, Birmingham, 1986.

[77] P. Hajela. Genetic algorithms in automated structural synthesis. In B. H. V. Topping, editor, *Optimization and artificial intelligence in civil and structural engineering*, volume 1, Dodrecht, 1992. Kluwer Academic Press.

[78] P. Hajela. Genetic search - an approach to the nonconvex optimization problem. *AIAA J*, 1990.

[79] K. Deb. Optimal design of a welded beam via genetic algorithms. *AIAA J*, 29:2013–15, 1991.

[80] O. König. *Evolutionary Design Optimization: Tools and Applications.* PhD thesis, Swiss Federal Institute of Technology, Zürich, 2004. Diss. ETH No. 15486.

[81] Peter J. Bentley, editor. *Evolutionary design by computers.* Morgan Kaufmann Publishers, Inc, San Francisco California, 1999.

[82] A. E. Nix and M. D. Vose. Modeling genetic algorithms with Markov chains. *Annals of Mathematics and Artificial Intelligence*, 5(1):79–88, 1992.

[83] G. I. N. Rozvany. Aims, scope basic concepts and methods of topology optimization. In G. I. N. Rozvany, editor, *Topology optimization in structural mechanics*, CISM courses and lectures - No.374, pages 1–55. Springer-Verlag Wien New York, 1997.

[84] Agoston Endre Eiben, Robert Hinterding, and Zbigniew Michalewicz. Parameter control in evolutionary algorithms. *IEEE Trans. on Evolutionary Computation*, 3(2):124–141, 1999.

[85] L. Kallel and M. Schoenauer. Fitness distance correlation for variable length representations, 1996. Technical Report 363, CMAP, Ecole Polytechnique.

[86] M. Gen and R. Cheng, editors. *Genetic algorithms and engineering optimization.* Wiley, New York, 2000.

[87] Bryant A. Julstrom. Comparing darwinian, baldwinian, and lamarckian search in a genetic algorithm for the 4-cycle problem. In Scott Brave and Annie S. Wu, editors, *Late Breaking Papers at the 1999 Genetic and Evolutionary Computation Conference*, pages 134 138, Orlando, Florida, USA, 13 1999.

[88] S.-S. Choi and B.-R. Moon. A graph-based lamarckian-baldwinian hybrid for the sorting network problem. *IEEE Transactions on Evolutionary Computation*, 9(1):105–14, 2005.

[89] Marc Wintermantel. *Design-Encoding for Evolutionary Algorithms in the Field of Structural Optimization.* PhD thesis, Swiss Federal Institute of Technology, Zürich, 2004. Diss. ETH No. 15323.

[90] T. Yu and P. Bentley. Methods to evolve legal phenotypes. In *Fifth International Conference on Parallel Problem Solving from Nature*, Amsterdam, 1998. Springer.

[91] M. Giger and P. Ermanni. Developement of CFRP racing motorcycle rims using a heuristic evolutionary algorithm approach. *Structural and Multidisciplinary Optimization*, 30(1):54–65, 2005.

[92] M. Giger and P. Ermanni. Evolutionary truss topology optimization using a graph-based parameterization concept. *Structural and Multidisciplinary Optimization*, 32(4):313 26, 2006.

[93] J.J. Merelo, M. Keijzer, and M. Schoenauer. EO, Evolutionary Computation Framework. http://eodev.sourceforge.net.

[94] M. Keijzer, J.J. Merelo, G. Romero, and M. Schoenauer. Evolving objects: a general purpose evolutionary computation library. In *EA-01, Evolution Artificielle, 5th International Conference in Evolutionary Algorithms*, pages 231–244, 2001.

[95] Matthew. GALib, Matthew's Genetic Algorithm Library. http://lancet.mit.edu/ga.

[96] D. Levine. PGAPack Parallel Genetic Algorithm Library. ftp://ftp.mcs.anl.gov/pub/pgapack.

[97] M. Giger. Strukturanalyse einer Formel-1-Felge. Term project IMES-ST-Nr 02-036., Center of Structure Technologies, Institute of Mechanical Systems, ETH Zürich, 2001.

[98] S.W. Tsai and E.M. Wu. A general theory of strength for anisotropic materials. *J. Composite Materials*, January:58 80, 1971.

[99] R. Manchek, J. Dongarra, and W. Jiang. PVM – Parallel Virtual Machine. http://www.csm.ornl.gov/pvm/pvm_home.html.

[100] M. Giger, D. Keller, and P. Ermanni. AORCEA - an Adaptive Operator Rate Controlled Evolutionary Algorithm. *Computers and Structures.* Submitted for publication.

[101] J. D. Schaffer, R. Caruana, l. Eshelman, and R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In J. D. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 51–60, Morgan Kaufmann, 1989.

[102] Peter J. Angeline. Adaptive and self-adaptive evolutionary computations. In Marimuthu Palaniswami and Yianni Attikiouzel, editors, *Computational Intelligence: A Dynamic Systems Perspective*, pages 152–163. IEEE Press, 1995.

[103] A. Tuson and P. Ross. Adapting operator settings in genetic algorithms. *Evolutionary Computation*, 6(2):161–184, 1998.

[104] L. Davis. Adapting operator probabilities in genetic algorithms. In J. J. Grefenstette, editor, *Proceedings of the 1st International Conference on Genetic Algorithms and their Applications*, pages 61–69. Lawrence Erlbaum Associates, 1989.

[105] Romanas Puisa. *The Adaptive Stochastic Algorithms for Structure Optimisation of Mechanical Parts.* PhD thesis, Vilnius Gediminas Technical University, Faculty of Mechanics, Department of Machine Building, 2005. UDK 531.8:51(043.3).

[106] D. Ackley. *A connectionist machine for genetic hill climbing.* Kluwer Academic Publishers, Boston, 1987.

[107] R. Thomsen and T. Krink. Self-adaptive operator scheduling using the religion-based EA. In J. J. Merelo, editor, *Parallel Problem Solving from Nature PPSN VII*, pages 214 223, Berlin, 2002. Springer.

[108] H. Muhlenbein and d. Schlierkamp-Voosen. Predictive models for breeder genetic algorithms. *Evolutionary Computation*, 1(1):25 49, 1993.

[109] C. A. Floudas and P. M. Pardalos. A collection of test problems for constrained global optimizaation algorithms. In *Lecture Notes in Computer Science*, volume 455. Springer Verlag, 1987.

[110] W. Hock and K. Schittkowski. Test examples for nonlineaar programming codes. In *Lecture Notes in Economics and Mathematical Systems*, volume 187. Springer Verlag, 1981.

[111] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1:80–83, 1945.

[112] D. Goldberg, K. Deb, and B. Korb. Messy genetic algorithms: motivation, analysis, and first results. *Complex Systems*, 3:493–530, 1989.

[113] H. Kargupta and S. Bandyopadhyay. A perspective on the foundation and evolution of the linkage learning genetic algorithms. *Computer methods in applied mechanics and engineering*, 186:269–294, 2000.

[114] I. Harvey. Species adaptation genetic algorithms: a basis for a continuing SAGA. In F.J. Varela and P. Bourgine, editors, *Toward a practice of autonomous systems, proceedings of first european conference on artificial life*, pages 346–54. MIT Press / Bradford Books, 1992.

[115] I. Harvey. The SAGA cross: the mechanics of recombination for species with variable-length genotypes. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature 2*, pages 269–78. North-Holland, 1992.

[116] S. Zebulum, M.A. Pacheco, and M. Vellasco. Variable length representation in evolutionary electronics. *Evolutionary Computation*, 8(1):93–120, 2000.

[117] I.Y. Kim and O.L. de Weck. Variable chromosome length genetic algorithm for progressive refinement in topology optimization. *Struct Multidisc Optim*, 29:445–56, 2005.

[118] J. Ryoo and P. Hajela. Handling variable string lengths in ga-based structural topology optimization. *Structural and Multidisciplinary Optimization*, 26:318–325, 2004.

[119] T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In L.J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–92. Morgan Kaufmann, 1995.

[120] Connie Loggia Ramsey, Kenneth A. De Jong, John J. Grefenstette, Annie S. Wu, and Donald S. Burke. Genome length as an evolutionary self-adaptation. *Lecture Notes in Computer Science*, 1498:345–54, 1998.

[121] N. Zehnder and P. Ermanni. A methodology for the global optimization of laminated composite structures. *Composite Structures*, 72:311–20, 2006.

[122] A.S. Wu and R.K. Lindsay. Empirical studies of the genetic algorithm with non-coding segments. *Evolutionary Computation*, 3(2):121–147, 1995.

[123] A.S. Wu and R.K. Lindsay. A comparison of the fixed and floating building block representation in the genetic algorithm. *Evolutionary Computation*, 4(2):169–193, 1996.

[124] C. Boesch. Evolutionäre Optimierung von Laminatstrukturen mittels variabler Genotyplänge. Semester thesis, ETH Zurich, Center for Structure Technologies, 2005. IMES-ST 05-107.

[125] X. Gou, G. Cheng, and K. Yamazaki. A new approach for the solution of singular optima in truss topology optimization with stress and local buckling constraints. *Structural and Multidisciplinary Optimization*, 22:364–372, 2001.

[126] D. Wang, W. H. Zhang, and J. S. Jiang. Combined shape and sizing optimization of truss structures. *Computational Mechanics*, 29:307–312, 2002.

[127] X. Gou, W. Liu, and H. Li. Simultaneous shape and topology optimization of truss under local and global stability constraints. *Acta Mechanica Solida Sinica*, 16(2), 2003.

[128] N. L. Pedersen and A. K. Nielsen. Optimization of practical trusses with constraints on eigenfrequencies, displacements, stresses, and buckling. *Structural and Multidisciplinary Optimization*, 25(5-6):436–445, 2003.

[129] G. Cheng and X. Guo. ϵ-relaxed approach in structural topology optimization. *Structural and Multidisciplinary Optimization*, 13:258–266, 1997.

[130] G. I. N. Rozvany. On design-dependent constraints and singular topologies. *Structural and Multidisciplinary Optimization*, 21:164–172, 2003.

[131] Y. M. Xie and G. P. Steven. *Evolutionary Structural Optimization*. Springer-Verlag, 1997.

[132] G. I. N. Rozvany. Stress ratio and compliance based methods in topology optimization - a critical review. *Structural and Multidisciplinary Optimization*, 21:109–119, 2001.

[133] W. Lingyun, Z. Mei, W. Guangming, and M. Guang. Truss optimization on shape and sizing with frequency constraints based on genetic algorithm. *Computational Mechanics*, published online 17 November 2004, 2004.

[134] H. Kawamura, H. Ohmori, and N. Kito. Truss topology optimization by a modified genetic algorithm. *Structural and Multidisciplinary Optimization*, 23:467–472, 2002.

[135] A. Kawamoto, M. P. Bendsøe, and O. Sigmund. Planar articulated mechanism design by graph theoretical enumeration. *Structural and Multidisciplinary Optimization*, 27:295–299, 2004.

[136] R. Balakrishnan and K. Ranganathan. *A textbook of graph theory.* Springer-Verlag, New York, Berlin, Heidelberg, 2000.

[137] J.G. Siek, L.Q. Lee, and A. Lumsdaine. *The Boost Graph Library.* C++ In-Depth Series. Addison Wesley, 2002.

[138] P. W. Fowler and S. D. Guest. A symmetry extension of maxwell's rule for rigidity of frames. *International Journal of Solids and Structures*, 37:1793–1804, 2000.

[139] Boost libraries. http://boost.org.

[140] M. Stolpe and K. Svanberg. A note on stress-constrained truss topology optimization. *Structural and Multidisciplinary Optimization*, 25:62 64, 2003.

[141] M. Pyrz. Evolutionary algorithm integrating stress heuristics for truss optimization. *Optimization and Engineering*, 5:45 -57, 2004.

[142] S.-J. Wu and P.-T. Chow. Steady-state genetic algorithms for discrete optimization of trusses. *Computers & Structures*, 56:979–991, 1995.

[143] A. C. C. Lemonge. *Application of Genetic Algorithms in Structural Optimization Problems.* PhD thesis, Federal University of Rio de Janeiro, Brazil, 1999. Program of Civil Engineering-COOPE.

[144] A. C. C. Lemonge and H. J. C. Barbosa. An adaptive penalty scheme for genetic algorithms in structural optimization. *Int. J. Numer. Meth. Engng*, 59:703–736, 2004.

[145] P. E. Gill, W. Murray, and M. A. Sanders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM J. Optim*, 12:979–1006, 2002.

[146] M. Giger, D. Keller, and P. Ermanni. A graph-based parameterization concept for global laminate optimization. *Structural and Multidisciplinary Optimization.* Submitted for publication.

[147] R. G. Le Riche, C. Knopf-Lenoir, and R. T. Haftka. A segregated genetic algorithm for constrained structural optimization. In Larry Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 558–65, San Francisco, CA, 1995. Morgan Kaufmann.

[148] L. Grosset, S. Venkataraman, and R. T. Haftka. Genetic optimization of two-material composite laminates, 2001. citeseer.ist.psu.edu/590751.html.

[149] O.C. Zienkiewicz and R.L. Taylor. *The finite element method*, volume 2. Oxford, 5 edition, 2000.

[150] O. König, M. Wintermantel, and N. Zehnder. The finite element library experiment. http://felyx.sourceforge.net.

[151] Denis Howe. *Aircraft Loading and Structural Layout*. Aerospace Series. Professional Engineering Publishing Limited, 2004.

[152] R. M. Jones. *Mechanics of Composite Materials*. Taylor & Francis Inc., 2 edition, 1999.

[153] R. Hill. *The Mathematical Theory of Plasticity*. Oxford University Press, London, 1950.

[154] Z. Hashin. Failure criteria for unidirectional composites. *Journal of applied mechanics*, 47:329 334, 1980.

[155] P. Ermanni and G. Kress. *Leichtbau III: Faserverbundstrukturen*. Centre of Struture Technologies, ETHZ, 2004.

[156] C. Ledermann. *Parametric Associative CAE Methods in Preliminary Aircraft Design*. PhD thesis, Swiss Federal Institute of Technology, Zürich, 2006. Diss. ETH No. 16778.

[157] R.D. Cook, D.S. Malkus, M.E. Plesha, and R.J. Witt. *Concepts and applications of finite element analysis*. John Wiley and Sons, fourth edition, 2002.

[158] Silicon Graphics Inc. STL - Standard Template Library. http://www.sgi.com/tech/stl.

[159] O. Schanke and K. Gärtner. Solving unsymmetric sparse systems of linear equations with pardiso. *Journal of Future Generation Computer Systems*, 20(3):475–87, 2004.

[160] Roger G. Grimes, John G. Lewis, and Horst D. Simon. A shifted block Lanczos algorithm for solving sparse symmetric generalized eigenproblems. *SIAM Journal on Matrix Analysis and Applications*, 15(1):228–272, 1994.

[161] L. R. Foulds. *Graph theory applications*. Springer-Verlag, New York, Berlin, Heidelberg, 1994.

[162] W.-K. Chen. *Graph theory and its engineering applications*, volume 5. World Scientific Publishing Co. Pte. Ltd., P O Box 128, Farrer Road, Singapore 912805, 1997.

[163] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *ACM Proceedings ot the 24th National Conference.* Association of Computing Machinery, New York, 1969.

Own publications

[1] M. Giger, D. Keller, and P. Ermanni, A graph-based parameterization concept for global laminate optimization, *Submitted to Structural and Multidisciplinary Optimization*, 2006.

[2] M. Giger, D. Keller, and P. Ermanni. AORCEA - an Adaptive Operator Rate Controlled Evolutionary Algorithm. *Submitted to Computers & Structures*, 2006.

[3] M. Giger and P. Ermanni. Evolutionary truss topology optimization using a graph-based parameterization concept. *Structural and Multidisciplinary Optimization*, 32(4):313-26, 2006.

[4] M. Giger and P. Ermanni. Development of CFRP racing motorcycle rims using a heuristic evolutionary algorithm approach. *Structural and Multidisciplinary Optimization*, 30(1):54-65, 2005.

[5] M. Redhe, M. Giger, and L. Nilsson. An investigation of structural optimization in crashworthiness design using a stochastic approach. *Structural and Multidisciplinary Optimization*, 27(6):44659, 2004.

[6] N. Stander, W. Roux, M. Giger, M. Redhe, N. Fedorova, J. Haarhoff. A Comparison of Metamodeling Techniques for Crashworthiness Optimization. *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, New York, 2004.

[7] N. Stander, W. Roux, M. Giger, M. Redhe, N. Fedorova, J. Haarhoff. Case Studies in Metamodeling and Random Search Techniques. *Proceedings of the European LS-DYNA Users Conference*, Ulm, Germany, 2003.

Curriculum Vitae

Name: Mathias Giger
Date of Birth: August 31 1977
Nationality: Switzerland
Address: St.Gallerstrasse 42,
CH-7320 Sargans

EDUCATION

Ph.D., Centre of Structure Technologies, *ETH Zurich* Apr03 – Jan07

Diploma thesis, *University of Linkoping*, Sweden Oct02 – Mar03
Division of Solid Mechanics, Dept of Mechanical Engineering

Study Mechanical Engineering, *ETH Zurich* Oct97 – Oct02
Subjects:
- Aerospace and lightweight structures
- Composite structures
- Product development

WORKING EXPERIENCE

25% employment, EVEN AG, Zurich since Sep05
Mechanical engineering, evolutionary optimization

Scientific assistant, *ETH Zurich* Mar02 – Oct02
Centre of Structure Technologies
Responsible for FEM lectures

Internship, *Contraves Space*, Oerlikon Jul01 – Sep01
Stress Office, W-ES2
Development of a computer program for dimensioning bolted joints

Internship, *Tribecraft AG*, Zurich Dec99 – Jun00
Product development