Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Master Thesis

Sebastian Mau

Topology Optimization using Column Generation Methods

Fakultät Technik und Informatik
Department Maschinenbau und Produktion

Faculty of Engineering and Computer Science
Department of Mechanical Engineering and Production Management

Sebastian Mau

Topology Optimization using Column Generation Methods

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Berechnung und Simulation im Maschinenbau
am Department Maschinenbau und Produktion
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Erstprüfer: Prof. Ivo Nowak
Zweitprüfer/in : Prof. Jens Telgkamp

Abgabedatum: 15.03.2022

Abstract

This thesis presents a new decomposition-based approach for solving topology optimization problems and determines its feasibility and usefulness. Topology optimization problems provide the possibility to reduce the mass of a structural element with respect to a given load case by determining an optimal shape. In general, the related optimization problems are non-convex and difficult to solve. Most conventional approaches for solving such topology optimization problems are based on heuristics and solve the problems locally. The aim of solving a topology optimization problem globally motivates to develop a new approach to improve the quality of the results. The approach uses methods of the column generation algorithm, i.e. it solves a convex linear master problem repeatably instead of the complex original topology optimization problem. This thesis shows that it is possible to estimate a global solution point with a true lower bound in accordance to an original topology optimization problem and approximate the original problem by that master problem by using solutions of corresponding sub-problems. Furthermore, the implementation of the new approach for the solver "Decogo" using "Python" language is outlined.

In dieser Arbeit wird ein neuer dekompositionsbasierter Ansatz zur Lösung von Topologieoptimierungsproblemen vorgestellt und seine Machbarkeit und Nützlichkeit bestimmt. Topologieoptimierungsprobleme bieten die Möglichkeit, die Masse eines Strukturelements im Hinblick auf einen bestimmten Lastfall zu reduzieren, indem eine optimale Form bestimmt wird. Im Allgemeinen sind die entsprechenden Optimierungsprobleme nicht konvex und schwer zu lösen. Die meisten herkömmlichen Ansätze zur Lösung solcher Topologieoptimierungsprobleme basieren auf Heuristiken und lösen die Probleme lokal. Das Ziel, ein Topologieoptimierungsproblem global zu lösen, motiviert zur Entwicklung eines neuen Ansatzes, um die Qualität der Ergebnisse zu verbessern. Der Ansatz verwendet Methoden des Spaltengenerierungsalgorithmus, d.h. er löst immer wieder ein konvexes lineares Masterproblem anstelle des komplexen ursprünglichen Topologieoptimierungsproblems. Diese Arbeit zeigt, dass es möglich ist, einen globalen Lösungspunkt mit einer echten unteren Schranke in Übereinstimmung mit einem ursprünglichen Topologieoptimierungsproblem zu schätzen und das ursprüngliche Problem durch dieses Hauptproblem zu approximieren, indem Lösungen entsprechender Teilprobleme verwendet werden. Darüber hinaus wird die Implementierung des neuen Ansatzes für den Solver "Decogo" in der Sprache "Python" beschrieben.

Contents

1 Introduction **1**
1.1 Motivation and State of the Art 2
1.1.1 Multistart Behavior 2
1.2 Basic Idea 3
1.2.1 Reformulation 4
1.2.2 Master Problem 4
1.2.3 Sub-Problems 4

2 Topology Optimization Problem **6**
2.1 General TO Problem 6
2.1.1 Design Variables 6
2.1.2 Objective Function 7
2.1.3 Elasticity Problem Constraint 7
2.1.4 Volume Constraint 7
2.1.5 Formulation 8
2.2 Reformulation of a TO Problem 9
2.2.1 Simple Example 9
2.2.2 Block Definition 12
2.2.3 Block-separated Force Vector 14
2.2.4 Objective Function 14
2.2.5 Copy Constraints 14
2.2.6 Volume Constraint 15
2.2.7 Elasticity Problem Constraint 15
2.2.8 Active and Passive Elements 15
2.2.9 Block-Separated TO Problem 17
2.2.10 MINLP Block-Separated TO Problem 17

3 FEM Formulation **18**
3.1 Symbols of Domain Definition 19
3.2 General Properties 20
3.2.1 `Node` Object 20
3.3 Finite Element Analysis 20
3.3.1 Elasticity Problem 20
3.3.2 Element Formulation 21
3.3.3 `Element` Object 24
3.3.4 Global Stiffness Matrix 25
3.4 Boundary Conditions 25
3.4.1 Fixations 25
3.4.2 Load Cases 26

3.4.3 BoundaryConditions Object 26
3.5 Solving Elasticity Problems 27
3.5.1 Incorporating Boundary Conditions 27
3.5.2 Direct 28
3.5.3 Iterative 28
3.6 FEModel Object 29
3.6.1 Constructor 30
3.6.2 Attributes 30
3.6.3 Methods 31
3.7 Experimental Model 35
3.7.1 Mapping Node to Degree of Freedom 36
3.7.2 Global Force Vector 37
3.8 Domain Decomposition of TO-Problem 38
3.8.1 Sub-Domain Definition 39
3.8.2 Sub-Domain's FEA 40
3.8.3 SubDomain Object 41
3.8.4 Boundary Definition 41
3.8.5 Boundary Forces 41
3.8.6 Incorporate Boundary Forces to Objective Function . 44
3.8.7 TODecomposer Object 48

4 Approaches for Solving a TO-Problem 50
4.1 OC - Optimality Criterion 50
4.1.1 SIMP - Solid Isotropic Material with Penalization . . . 51
4.1.2 RAMP - Rational Approximation with Material Properties 52
4.2 SKO - Soft-Kill-Option 53
4.3 TOSS - Topology Optimization for Stiffness and Stress 53
4.4 Filter 53
4.5 Solving TO Problem with SIMP Approach 56
4.6 Algorithms using SIMP Approach 57
4.7 SIMPProblem Object 60
4.7.1 Constructor 60
4.7.2 Attributes 60
4.7.3 Function solve 62
4.7.4 Function __oc__ 63
4.8 Results for Experimental Model 64
4.8.1 Configuration and Start Conditions 64
4.8.2 Results 64
4.8.3 Validation 66

5 Column Generation for TO Problems **67**
5.1 Classic Column Generation 67
5.2 Apply CG on Block TO 68
5.3 Sub-Problems 70
5.3.1 Algorithm 71
5.4 TOCG algorithm 72

6 Implementation and Results **73**
6.1 Basic Course of Action 73
6.1.1 Implementation Structure 73
6.2 `TOModelBase` Object 74
6.2.1 Instantiate 74
6.2.2 Attributes 75
6.2.3 Methods 75
6.2.4 `TOReformulatedModel` object 76
6.2.5 `TODecomposer` class 76
6.3 Model Creation 76
6.3.1 Programmatic 77
6.3.2 Using "Gmsh" 77
6.3.3 `MSHParser` class 78
6.4 Decogo 79
6.4.1 Generic Framework of Decogo 79
6.5 Decogo TO Layer 82
6.5.1 Input Model 82
6.5.2 CG Cut Pool 83
6.5.3 Sub-Model 84
6.5.4 Sub-Problem 84
6.5.5 Original-Problem 85
6.5.6 Algorithm 87
6.6 Solving TO Problems using Decogo 88
6.6.1 NLP 89
6.6.2 MINLP 92
6.7 Extensions 96
6.7.1 Pertubation Extension 96
6.7.2 Direction Stabilizer Extension 101

7 Conclusion **102**

8 Prospect **105**
8.1 Reaching Global Optimality 105
8.2 Locally Solving Sub-Problems 106

8.3 Impact of Volume Constraint 106
8.4 Implementation 107

9 Additional and Prepared Features 108

A Annex I
A.1 Folder Structure I
A.2 Programmatic Model III
A.3 Mesh File V
A.4 Configuration Dictionary VII
A.5 Log Files VIII
A.5.1 NLP Variant VIII
A.5.2 MINLP Variant XI
A.5.3 NLP Variant with Column Pertubation XVIII
A.5.4 MINLP Variant with Column Pertubation XXI

List of Figures

1 Multistart investigation for classic SIMP-Approach with 100 runs . . . 3
2 Simple Example for Reformulation . . . 9
3 Simple Example for Reformulation . . . 10
4 Visualization of used finite elements for topology optimization in this thesis . . . 22
5 Experimental Model . . . 35
6 Cutting Lines for Sample Main-Domain . . . 39
7 Idea of Boundary Forces . . . 43
8 Model to prove $\vec{g} = \vec{0}$ . . . 46
9 Block Model . . . 46
10 Effect of Penalization Factor for SIMP Approach . . . 51
11 Effect of Penalization Factor for RAMP Approach . . . 52
12 Possible unwanted Checkerboard Effect . . . 54
13 Results of the TO problem of the experimental model using standard SIMP approach . . . 65
14 General course of action of the implementation to solve a TO problem with new approach . . . 74
15 Geometry and mesh created using "Gmsh" . . . 77
16 Export options . . . 78
17 UML diagram of framework . . . 81
18 Resulting density distributions . . . 91
19 Resulting density distributions of MINLP TO problem . . . 95
20 Objective value of MP for NLP variant . . . 98
21 Density distributions for column pertubation . . . 99
22 Density distributions for MINLP variant with pertubation extension . . . 100

List of Tables

1 Symbols for general Topology Optimization Problem 19
2 Most Important Attributes of FEModel class 31
3 Basic Properties of a sub-domain Ω_k 40
4 Mutable Parameters 57
5 Attributes of `SIMPProblem` class 61
6 Abstract classes of framework 80
7 First results for experimental model solved by Decogo 90
8 Results for experimental model solved by Decogo as MINLP problem 94

1 Introduction

Saving energy and material are some of the main challenges to handle the climate crises. Besides exploring and developing climate-neutral technologies, the reduction of a structure's weight is a key to solving these tasks. For example, the energy consumption of a means of transport for moving depends strongly on its weight. The research field of topology optimization (TO) deals with the mass reduction of a given geometry, or rather finding a geometry's optimal shape in accordance to a given load case. This thesis develops a basis for a new generate-and-solve approach for topology optimization problems and tries to provide a starting point for further research. One goal of this report is checking the feasibility and usefulness of using column generation (CG) methods for solving TO problems. Another goal is developing a topology optimization layer for the existing solver "Decogo", which was developed at the university of applied science in Hamburg (HAW Hamburg), to apply the new topology optimization approach. The solver is completely written in the Python language. During this report, using a simple experimental model (3.7) shows the basic course of action and visualizes the theory of the new method. The implementation can be found in the "GitHub" repository "aWsKlixz/DecogoTOLayer" [Tol], including some examples. The most important aspects are also provided in the Annex. While describing the new approach, some implementation details are outlined.
The report starts with clarifying the motivation as to why this new approach could be interesting and outlines the current state of the art for solving TO problems (1.1). In addition, the basic idea of the new method is highlighted (1.2). Section 2 presents the definition of a general TO problem based on reducing a geometry's compliance, including a corresponding block-separable reformulation. After that, section 3 defines a general domain for TO problem, its decomposition and the related finite element analysis (FEA). Furthermore, this section highlights the experimental model which is used for the application and visualization of the new approach. The next section (4) deals with different conventional approaches to solve a TO problem. Following, in section 5, the column generation algorithm is explained and applied to TO problems. Further, the corresponding algorithms are presented to solve a TO problem using column generation methods. Section 6 outlines the implementation details and presents first results for the new approach. A conclusion then summarizes the use of CG for TO problems (7) before a prospect for further investigations is provided (8). Lastly, some additional features of the implementation are presented briefly (9).

1.1 Motivation and State of the Art

As long as, a general mixed-integer quadratic (MIQP) topology optimization problem is small, it can be solved using global solving approaches, e.g. a branch-and-bound algorithm. Application-oriented TO problems are very large, so that the global solving methods do not find the corresponding global solution in a reasonable time. Most of the existing solving methods use heuristic approaches with strong simplifications to avoid this issue. One of the most common simplifications is a transfer of a mixed-integer quadratic (MIQP) TO problem to a none-linear programming problem (NLP) by means of connecting properties continuously with empirical laws instead of proved mathematical connections, so that a TO problem is solved locally instead of globally. As the existing approaches are heuristics and solve a TO problem locally only, these approaches strongly depend on initial values for solving. Although, TO problems are only solved locally, experience has shown that reasonable results can be achieved in realistic applications in reasonable calculation time. This thesis and the new approach is motivated by several challenges of TO problems. Firstly, the quality should be improved, or rather the heuristic character of the existing approaches should be decreased, regarding the results of a TO problem as the new method bases on a global optimization approach, so that it is possible to make a statement whether a result is globally optimal or whether there may be better solutions. Furthermore, it can be assumed that it is possible to consider large TO problems as true mixed-integer optimization problems and the presented approach is assumed to create the opportunity to solve complex TO problems fast by domain decomposition and parallel solving sub-problems.

1.1.1 Multistart Behavior

This section outlines briefly the meaning of the initial values dependency for a TO problem by means of a classic SIMP approach (4.1.1) without going into detail. The results for a multi-start investigations can be found in the figure below.

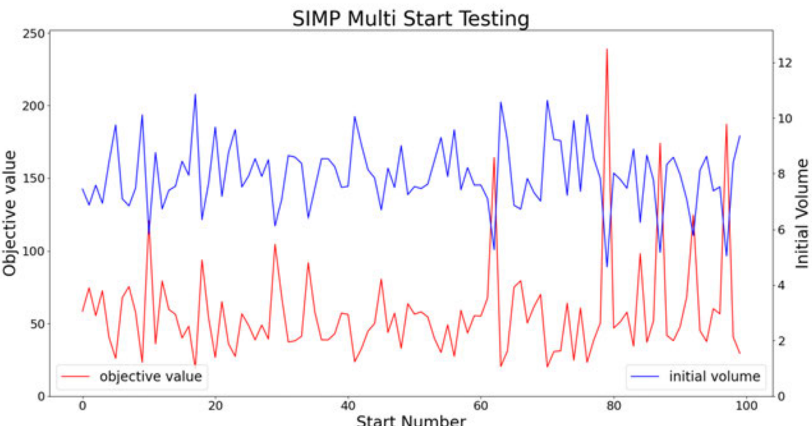


Figure 1: Multistart investigation for classic SIMP-Approach with 100 runs

For this chart, the experimental TO model, which is presented later, is solved 100 times with random initial values using a classic SIMP approach. As shown, the resulting objective value strongly depends on the initial values. An approach to reduce this effect (8.3) does exist, but the scale of this report is restricted and focuses more on developing a working prototype of the new approach. This is just an extreme example for visualizing the heuristic character of a conventional approach and showing that the TO problem is solved locally.

1.2 Basic Idea

The basic idea of this new decomposition based approach is reformulating the TO problem to a block-separable problem, based on domain decomposition. Using that block-separable problem, a linear master problem (MP) is formulated that can be efficiently solved globally, e.g. by using the Simplex algorithm. The MP describes an approximation of the original problem (OP) and depends on solutions of sub-problems of the block-separable problem. By solving the MP and the sub-problems alternately, the quality of the approximation can be increased. In the end, the solution of the MP provides an estimated value of the global solution point of the original problem that can be used for further searching the global optimum of that original problem. As the MP is much easier to solve, a global solver without any heuristics is used to determine an estimated global solution candidate, the dual bound, related to the OP. Also, the sub-problems are small, making it possible to solve them globally. The process briefly described above for solving a complex problem with the help of a linear master problem is the basis of column

generation (CG) algorithms. More information about CG and applying it to a TO problem is presented in section 5.

1.2.1 Reformulation

At first, the original TO problem needs to be reformulated to a block-separable problem so that the CG method can be applied to the problem. Regarding the TO problem, the domain of the original problem will be decomposed in smaller sub-domains, whereby each defines a smaller TO problem itself and corresponds to a block of the block-separable reformulation. The blocks will be independent of each other, so they can be solved separately. By adding global linear constraints, the blocks are connected again. The reformulation of the TO problem is shown in section 2.2.

1.2.2 Master Problem

The master problem defines a relaxation of the convex hull of the OP. After applying the new approach, the master problem provides a true lower bound (dual bound) and an estimated solution of the OP. This true dual bound of the TO problem is the innovation of this new approach. The estimated solution point can be used as a starting point for further searching the global solution of the OP. In addition, the dual values of the MP define the objective of the sub-problems. Compared to the original problem, the size of the MP is very small, so that it is time- and performance-effective to solve the MP as often as required. For updating the MP and improving the quality of the approximation related to the original problem, results of sub-problems are used, as the sub-problems depend on dual values of the MP. The objective value of the master problem defines an approximated dual bound of the original problem as it is solved globally. A local solver for the TO problem can be used to determine the primal bound. The gap between the dual and primal bound should be as small as possible so that the global solution of the original problem can be found as easily as possible. As the MP is just an approximation of the original problem, the gap can not be closed. In this report, the solver "Gurobi"[Gur] is used to solve the MP.

1.2.3 Sub-Problems

Related to the block-separable formulation, the sub-problems of those blocks are also TO problems themselves, but they are much smaller than the original TO problem. The size of these sub-problems can be controlled and needs to be sufficiently small to solve the sub-problems globally. As this thesis

investigates the feasibility and usefulness of this new approach, an existing conventional solver is used instead of developing a global solver for TO sub-problems as this would unfortunately exceed the scope of this report. Nevertheless, it is highly recommended to develop a global solver to improve the results of the sub-problems, as they define the quality of the approximation through the master problem. Sub-problems are solved very often during the new process. The algorithm for solving TO sub-problems is outlined in section 5.3.1. An advantage of this new approach is the possible usage of any topology optimization solver to determine local and global solutions of the sub-problems as this new method is fully generic. Some conventional approaches for solving TO problems are outlined in chapter 4.

2 Topology Optimization Problem

This section starts with presenting a general formulation for a topology optimization problem (2.1.5), including the properties which define a TO problem (3.2), the objective function (2.1.2), and the required constraints (2.1.4). Furthermore, the reformulation of a TO problem into a block-separable form is outlined in section 2.2. For an easier understanding, the reformulation is applied on a simple example (2.2.1) before its general character is explained.

2.1 General TO Problem

Topology optimization problems try to find an optimal design shape for a given load case and base on the finite element method (FEM) approximation. The geometry to be optimized is meshed with finite elements and normalized densities are assigned to each element ρ_e. During the optimization, the densities change between zero and one so that this process determines areas of geometry where it is possible to reduce the geometry's mass by removing elements with a density equal to zero and calculates an optimal density distribution for each element for the whole geometry.
The objective of this general TO problem minimizes the compliance c, which is the reciprocal value of the stiffness, of a geometry. Minimizing the compliance is the same as maximizing the stiffness. The optimization variables or rather the design variables are the displacements of the system's nodes and the densities of the different elements.
As this report also considers domain decomposition for topology optimization problems, the original complete finite element system is also called main-domain Ω. In the following sections, properties, the objective function and constraints will be defined which are required for a general topology optimization problem.

2.1.1 Design Variables

TO problems presented in this report contain two design variables, one for the densities and the other one for the displacements of the degrees of freedom at the model's nodes. The vector $\underline{\rho}$ denotes the densities, which equal zero or one for a mixed-integer optimization problem. The vector $\vec{u}$ maps the displacements and has a direction in the design space, unlike the density vector. Both design variables can be summarized into the vector

$$\underline{x} = (\underline{\rho}, \vec{u}) \tag{1}$$

2.1.2 Objective Function

The topology optimization problems considered in this report minimize the compliance c of a domain, which is defined by

$$C := \vec{u}^T \boldsymbol{S}(\underline{\rho})\vec{u}, \tag{2}$$

where $\boldsymbol{S}(\underline{\rho})$ denotes the global stiffness matrix depending on the densities $\underline{\rho}$ of the domain Ω. The objective is quadratic and depends on mixed-integer variables, so it is not easy to solve with conventional global solving approaches. Using the force vector $\vec{f}$, which maps the constant load case of a TO problem, and the elasticity problem (4), it is possible to write the function as

$$c = \vec{f}^T \vec{u}. \tag{3}$$

This form matches the required form for the CG algorithm. The equation shows that the total compliance is the sum of single compliances related to the corresponding displacement. The compliance become smaller for smaller displacements, as the forces are constants.

2.1.3 Elasticity Problem Constraint

The linear elasticity equation is simple defined as

$$\boldsymbol{S}\vec{u} = \vec{f} \tag{4}$$

for the complete domain. In a TO problem, this equation is used as a constraint that needs to be fulfilled at any time of the optimization. The equation is solved during a finite element analysis (FEA) to receive the displacements at the nodes during the optimization process. In addition, the global stiffness matrix is calculated by means of the elements' stiffness matrices as

$$\boldsymbol{S}(\underline{\rho}) = \sum_{e \in M} \rho_e \cdot \boldsymbol{S}_e, \tag{5}$$

including the general approach-specific dependency on the element's densities.

2.1.4 Volume Constraint

The complete volume of a domain depends on the elements' densities and is given by

$$V(\underline{\rho}) = \sum_{e \in M} \rho_e \cdot v_e, \tag{6}$$

where v_e denotes the volume of a finite element e and M represents the set of elements of the domain. TO problems in this report optimize the compliance of a domain related to a target volume

$$V_f = v_f \cdot V^0 \tag{7}$$

which should be reached using the optimization. V^0 maps the original volume of the domain, where the elements densities equal one and the factor v_f denotes a percentage of the original volume. This factor is defined by the user and is within in a range between zero and one. The volume constraint is given by

$$\frac{V(\underline{\rho})}{V^0} \leq v_f. \tag{8}$$

Sub-gradient OC approaches require the volume constraint to receive a convex optimization problem so that the optimality criterion for the corresponding Lagrangian equation can be determined.

2.1.5 Formulation

What follows is the formulation of a general topology optimization problem.

$$\begin{aligned}
min \quad & \vec{f}^T \vec{u} \\
s.t. \quad & \boldsymbol{S}(\underline{\rho})\vec{u} = \vec{f} \\
& \frac{V(\underline{\rho})}{V_0} \leq v_f \\
& \rho_i \in \{0,1\}, \quad i \in M \\
& u_d \in \left[u_d^-, u_d^+\right], \quad d \in D \\
& \boldsymbol{S}(\underline{\rho}) = \sum_{i \in M} \rho_i \cdot \boldsymbol{S}_i,
\end{aligned}$$

where M denotes the set of finite elements and D maps the displacements. To solve this problem effectively, several approaches exists, which are presented in chapter 4.

2.2 Reformulation of a TO Problem

This section deals with the reformulation of the original TO problem regarding the required block-separable form, including the block definition for a TO problem. In addition to the reformulation of the objective function, the elasticity problem constraint and the volume constraint, the connection of the sub-domains is outlined too. In the end, this section shows the complete reformulated topology optimization problem (RTO), which is the basis to apply column generation methods for topology optimization.

2.2.1 Simple Example

The reformulation of the general original TO problem is shown by means of a very simple one dimensional example made of two truss elements (figure 2).

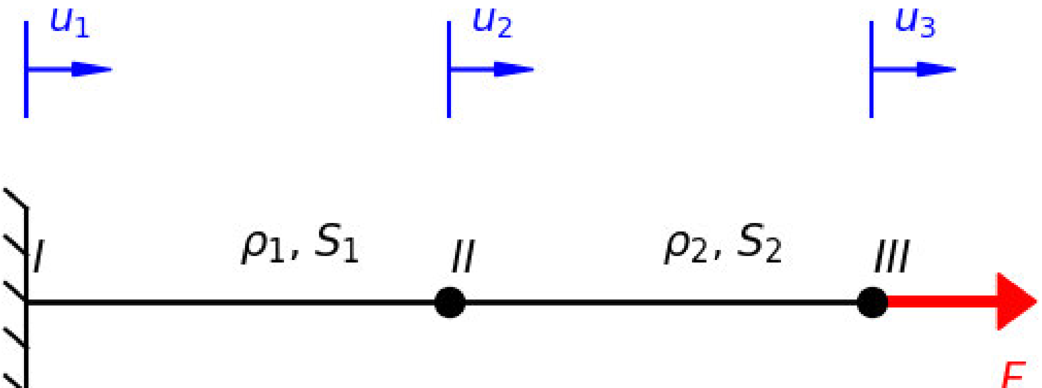


Figure 2: Simple Example for Reformulation

Transferring on the general formulation from section 2.1.5, the corre-

sponding simplified general TO problem is given by

$$\begin{aligned} min \quad & \vec{f}^T \vec{u} \\ s.t. \quad & \boldsymbol{S}\vec{u} = \vec{f} \\ & \frac{V(\underline{\rho})}{V_0} \leq v_f \\ & \rho_i \in \{0,1\}, \;\; i \in M \\ & u_d \in \left[u_d^-, u_d^+\right], \;\; d \in D \\ & M = \{1,2\}, \;\; D = \{1,2,3\} \\ & \underline{\rho} = \begin{pmatrix} \rho_1 \\ \rho_2 \end{pmatrix}, \;\; \vec{u} = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \\ & \boldsymbol{S} = \begin{pmatrix} S_1 & -S_1 & 0 \\ -S_1 & S_1 + S_2 & -S_2 \\ 0 & -S_2 & S_2 \end{pmatrix}. \end{aligned}$$

The dependency of the density variables are removed from the stiffness matrix to reduce the complexity from the example. This model is split into two sub-models by cutting node II so that the following block-model, made of two blocks $K = \{1,2\}$, is created.

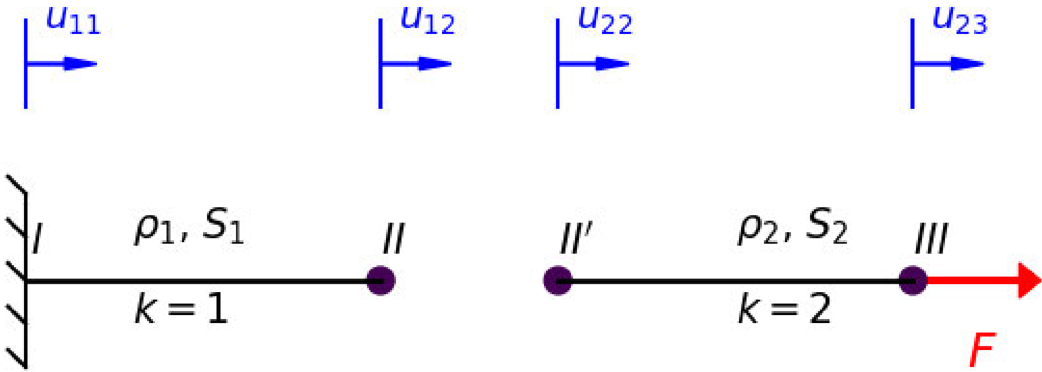


Figure 3: Simple Example for Reformulation

As shown in the figure above, the number of displacement variables is in-

creased to four. The stiffness matrix $\boldsymbol{S}$ of this new multi-model is given by

$$\boldsymbol{S} = \begin{pmatrix} S_1 & -S_1 & 0 & 0 \\ -S_1 & S_1 & 0 & 0 \\ 0 & 0 & S_2 & -S_2 \\ 0 & 0 & -S_2 & S_2 \end{pmatrix}.$$

The block diagonal structure can be disjointed into two separated blocks. To reach that separable stiffness matrix, node II is copied from the left block to the right block, i.e. II' denotes that copied node. To keep the connection between the blocks, so that the original problem is not changed, the new displacement of the copied node needs to be equal to the displacement of the original node, as

$$u_{22} = u_{12}.$$

This constraint is part of the new reformulation and is called "copy constraint". As this constraint contains variables from both blocks, it is a global constraint regarding the complete block-model. Below, the reformulation of this simple block-model is presented

$$\begin{aligned} min \quad & \vec{f}_1^T \vec{u}_1 + \vec{f}_2^T \vec{u}_2 \\ s.t. \quad & \left(\underline{\rho}, \vec{u}\right) \in P \quad \text{(Global Constraints)} \\ & \left(\underline{\rho}_k, \vec{u}_k\right) \in X_k, \quad k \in \{1, 2\} \quad \text{(Local Constraints)}, \end{aligned}$$

where

$$P = \{(\underline{\rho}, \vec{u}) \in \mathbb{Z}^2 \times \mathbb{R}^4 : u_{22} = u_{12}, \ \ V(\underline{\rho}) \leq v_f V_0\}$$

$$X_1 = \{\vec{u}_1 \in \mathbb{R}^2 : \boldsymbol{S}_1 \vec{u}_1 = \vec{f}_1\}$$

$$\boldsymbol{S}_1 = \begin{pmatrix} S_1 & -S_1 \\ -S_1 & S_1 \end{pmatrix}$$

$$\vec{u}_1 = \begin{pmatrix} u_{11} \\ u_{12} \end{pmatrix}$$

$$X_2 = \{\vec{u}_2 \in \mathbb{R}^2 : \boldsymbol{S}_2 \vec{u}_2 = \vec{f}_2\}$$

$$\boldsymbol{S}_2 = \begin{pmatrix} S_2 & -S_2 \\ -S_2 & S_2 \end{pmatrix}$$

$$\vec{u}_2 = \begin{pmatrix} u_{22} \\ u_{23} \end{pmatrix}$$

If a constraint contains variables from different blocks, it will be a global constraint. A local constraint just contains variables from one block. It should be noticed that the TO problem of the simple example does not lead reasonable solutions as no element can be omitted. The next sections describe the creation of sub-models and reformulation of the general TO problem in a general way.

2.2.2 Block Definition

The original TO problem is decomposed in $|K|$ blocks. The sub-model creation and sub-problem definition can be interpreted as cutting nodes of finite elements. The design variables for the densities and displacements of a block k are

$$\underline{\rho}_k = (\rho_{k,j}) \in \mathbb{Z}^{n_{ke}}, \ \ \rho_{k,j} \in \{0, 1\} \tag{9}$$

$$\vec{u}_k = (u_{k,j}) \in \mathbb{R}^{n_{kd}}, \ \ u_{k,j} \in \left[u_{k,j}^{+}, u_{k,j}^{-}\right], \tag{10}$$

where n_{ke} denotes the number of finite elements and n_{kd} is the number of possible displacements, or rather the number of degrees of freedom (DoF)

in a block k. In accordance to the block-separable problem, a summarized design variable for a block k is defined as

$$\underline{x}_k := (\underline{\rho}_k, \vec{u}_k) \tag{11}$$

with the corresponding dimension

$$n_k = |\underline{\rho}_k| + |\vec{u}_k|. \tag{12}$$

This leads to the requirement of the definition of indices sets for both variables relating to a summarized design variable $\underline{x}_k$. The index set for densities to the corresponding elements is defined as

$$[M_k] := \{0, \ldots, n_{ke} - 1\} \tag{13}$$

and the index set for the displacements is given by

$$[D_k] := \{n_{ke}, \ldots, n_{ke} + n_{kd} - 1\}. \tag{14}$$

Due to this index sets, it is still possible to differentiate between densities and displacements using the summarized variable $\underline{x}_k$.
The block-separable formulation means that each block needs to be independent of each other block, so that each block's problem can be considered by itself. This condition leads to the requirement that none variable must be in two different blocks so that

$$M_1 \cap M_2 \cap \cdots \cap M_{|K|} = \emptyset \tag{15}$$

$$D_1 \cap D_2 \cap \cdots \cap D_{|K|} = \emptyset. \tag{16}$$

This means in a more practical view, each element and node, i.e. the corresponding DoFs, needs to be in one block. Due to cutting nodes, this condition is already fulfilled for the elements so that

$$|M_1 \cap M_2 \cap \cdots \cap M_{|K|}| = 0 \tag{17}$$

and no adjustments have to be made. Regarding the nodes and consequently regarding the displacement variables, the problem has to be adjusted as

$$|D_1 \cap D_2 \cap \cdots \cap D_{|K|}| > 0. \tag{18}$$

The intersection of the different DoF sets is not empty and so displacement variables are used in multiple blocks. Displacements u_{ki} located on a block's boundary Γ_k are used in more than one block. The set D_Γ includes all displacement variables u_{ki} which are located on a boundary. To reach the block-separated formulation, boundary displacements are copied and assigned to

the related neighboring blocks. The set $\tilde{D}_k$ denotes the displacements of a block k including the copies. Those copied displacement variables are simply connected by their equality to each other. These connections, or rather constraints, are called copy constraints. The size of the original problem increases but it is necessary to reach the required block-separable formulation of the original problem.

2.2.3 Block-separated Force Vector

As long as forces only grip at nodes which are not cut, the original force vector $\vec{f}$ can be decomposed in terms of the corresponding displacements because the added variables do not impact the load cases. This changes if a node is cut where a force attacks: this node will be copied to each other domain and the force is distributed to the new nodes so that the sum of forces and therefore the original problem is not changed

$$\vec{f} = \sum_{k \in K} \vec{f}_k. \tag{19}$$

2.2.4 Objective Function

Before the copy constraints are defined explicitly, the objective function is considered. It defines the compliance of the domain depending on the load cases and the current displacements (equation 3). The transfer of the original formulation to the block-separable is given by

$$c = \vec{f}^T \vec{u} \iff \sum_{k \in K} \vec{f}_k^T \vec{u}_k. \tag{20}$$

2.2.5 Copy Constraints

For this step, the user selects where the main-domain is cut, so that the properties of the boundaries are indirectly defined by the user. The equality of the displacements on the boundaries of the different blocks determines the connection to each other. The set of copy constraints is defined by

$$I := \{\vec{u} : u_{kj} = u_{\ell i} \;\; \forall\, (k, j, \ell, i) \in \tilde{D}_\Gamma\}. \tag{21}$$

The number of copy constraints n_c depends on the number of the different DoFs on the boundaries.

2.2.6 Volume Constraint

The original volume constraint

$$\frac{V(\underline{\rho})}{V_0} \leq v_f \tag{22}$$

can be reformulated in the following block-separable form

$$\sum_{k \in K} V_k(\underline{\rho}_k) \leq v_f \cdot \sum_{k \in K} V_{k0} \tag{23}$$

$$\sum_{k \in K} \sum_{j \in M_k} v_{kj} \cdot \rho_{kj} \leq v_f \cdot \sum_{k \in K} V_{k0}. \tag{24}$$

This connection is reformulated further and leads to the following constraint formulation using the global density vector $\underline{\rho}$

$$\underline{v} := (v_i) \in \mathbb{R}^{n_e} \tag{25}$$

$$v_i := v_e, \ \ \forall e \in M \tag{26}$$

$$V := \left\{ \underline{\rho} \in \mathbb{R}^{n_e} : \underline{v}^T \underline{\rho} \leq v_f \cdot V_0 \right\}, \tag{27}$$

where the vector $\underline{v}$ contains the volumes of each element. Like the copy constraints, the volume constraint still depends on all blocks together.

2.2.7 Elasticity Problem Constraint

The elasticity problem constraint can be reformulated in a local block-specific constraint for a block k

$$X_k := \{\vec{u}_k \in \mathbb{R}^{n_{kd}}, \underline{\rho}_k \in \mathbb{R}^{n_{ke}} : \boldsymbol{S}_k(\underline{\rho}_k)\vec{u}_k - \vec{f}_k = 0\} \tag{28}$$

$$\boldsymbol{S}_k(\underline{\rho}_k) = \sum_{e \in M_k} \boldsymbol{S}_{ke} \cdot \rho_{ke}. \tag{29}$$

2.2.8 Active and Passive Elements

It is possible to consider active and passive elements in the TO problem. Both types depend on user definitions and boundary conditions of the main domain Ω. Active elements need a density of one, whereas passive elements need a density of zero. The user can define those elements to reach a specific

exterior shape or to ensure that a required fastening point exists. Boundary conditions can also define active elements, as an attached element to a loaded or fixed node needs to be active. Active and passive elements lead to definitions of relating sets and local constraints. The set M_a defines the active elements in a main-domain Ω, whereas the set M_p defines the passive elements. The corresponding block-specific constraints are

$$M_{ka} \subseteq M_a \subseteq M \tag{30}$$

$$M_{kp} \subseteq M_p \subseteq M. \tag{31}$$

The local constraints for the reformulation are defined as

$$J_k = \{\rho \subset \mathbb{Z} | \rho_{ke} = 1, \forall e \in M_{ka}\} \tag{32}$$

$$I_k = \{\rho \subset \mathbb{Z} | \rho_{ke} = 0, \forall e \in M_{kp}\}, \tag{33}$$

and can be appended to the corresponding block-specific constraints X_k as required.

2.2.9 Block-Separated TO Problem

This section outlines the complete reformulation of a TO problem to a block-separable formulation which is required to apply CG methods. The formulation differentiate between global and local block-separated constraints.

$$
\begin{aligned}
& min \;\; \sum_{k \in K} \vec{f}_k^T \vec{u}_k \\
& s.t. \;\; (\underline{\rho}, \vec{u}) \in P, \;\; (\underline{\rho}_k, \vec{u}_k) \in X_k, \;\; k \in K \\
& \qquad P = \left\{ \left(\underline{\rho}, \vec{u}\right) \in \mathbb{Z}^{|M|} \times \mathbb{R}^{|\tilde{D_\Gamma}|} : \underline{v}^T \underline{\rho} \le v_f \cdot V_0, \right. \\
& \qquad\qquad\qquad \left. u_{kj} = u_{\ell i} \;\; \forall \, (k, j, \ell, i) \in \tilde{D_\Gamma} \right\} \\
& \qquad X_k := \left\{ \left(\underline{\rho}_k, \vec{u}_k\right) \in \mathbb{R}^{n_{kd}} \times \mathbb{Z}^{n_{ke}} : \boldsymbol{S}_k \vec{u}_k - \vec{f}_k = 0 \right\}
\end{aligned}
\tag{34}
$$

The problem above is the basis for solving TO problems using CG methods. Later, some adjustments to this problem are presented. The following section deals with the explicit definition of a TO domain, its decomposition and finite element analysis.

2.2.10 MINLP Block-Separated TO Problem

Below, this sub-section briefly presents the formulation of the block-separated TO problem as a general MINLP formulation using the summarized variables $\underline{x}$ and $\underline{x}_k$.

$$
\begin{aligned}
& min \;\; \underline{c}^T \underline{x} \\
& s.t. \;\; \underline{x}_k \in X_k, \;\; \forall k \in K, \\
& \qquad \underline{x} \in P
\end{aligned}
\tag{35}
$$

The proportional factor c in the objective function is made of zeros and the block-specific force vectors $\vec{f}_k$, so that it corresponds to the objective of the block-separated TO problem (equation 20)

3 FEM Formulation

This section deals with the definition of a domain for the topology optimization problem. It starts by outlining the symbols and defining general properties (3.2) before the finite element analysis for a topology optimization domain is described (3.3). This includes the definition of the different available boundary conditions and methods for performing the FEA. After that, the explicit definition of an experimental model is shown on the basis of which the new approach for topology optimization problems and the corresponding implementation is described. Lastly, this chapter presents the applied domain decomposition, which is required regarding the block-separated TO problem. In between, brief descriptions of the implementation for corresponding objects methods and variables can be found. Further information about the implementation is provided in the repository.

3.1 Symbols of Domain Definition

The following table summarizes the properties and symbols which define a domain for a topology optimization problem.

Symbol	Name	Definition	Notes
n_e	Element Number	$n_e \in \mathbb{Z}; n_e > 0$	
n_n	Node Number	$n_n \in \mathbb{Z}; n_n > 0$	
n_{dn}	DoF Number per Node	$n_{dn} := 2$	Is constant in this report
n_d	DoF Number	$n_d \in \mathbb{Z}; n_d = n_{dn} \cdot n_n$	
M	Set of Elements	$M := \{1, 2, \dots, n_e\}$	
N	Set of Nodes	$N := \{1, 2, \dots, n_n\}$	
D	Set of DoFs	$D := \{1, 2, \dots, n_d\}$	$\lvert D \rvert = n_{dn} \cdot \lvert N \rvert$
N_f	Set of fixed Nodes	$N_f \subset N$	Is user-defined
D_f	Set of fixed Dofs	$D_f \subset D$	N_f defines this subset
$\vec{f}$	Force/ Load Vector	$\vec{f} := (f_d) \in \mathbb{R}^{n_d}$	Vector of user-defined outer forces
$\vec{u}$	Displacement Vector	$\vec{u} := (u_d) \in \mathbb{R}^{n_d}$	Vector for complete domain
$\boldsymbol{S}$	Stiffness Matrix	$\boldsymbol{S} := \sum_{e \in M} \boldsymbol{S}_e \in \mathbb{R}^{n_d \times n_d}$	Is assembled from elements stiffness matrices
V	Domain's Volume	$V := \sum_{e \in M} v_e$	Sum of Elements' Volumes

Table 1: Symbols for general Topology Optimization Problem

3.2 General Properties

Each geometry to be optimized, is meshed with finite elements so each domain Ω is made of a number of elements n_e which defines a set of elements

$$M := \{1, \ldots, n_e\} \tag{36}$$

and a number of nodes n_n which defines a set of nodes

$$N := \{1, \ldots, n_n\}. \tag{37}$$

During a finite element analysis, the displacements of the degrees of freedoms (DoF) of the nodes are considered. The number of degree of freedoms for a node n_{dn} or rather for a complete element depends on the dimension of the design space (2D or 3D) and the element formulation itself.

3.2.1 `Node` Object

Nodes of a domain are considered as an object in the implementation. For creating an instance of a node the global coordinates and a global node id are required. The `nid` property is used to identify a node clearly in the global system. Furthermore, the object contains the properties `x`, `y` and `z` to map the coordinates. While instantiating a node object, the value for the z-coordinate is an optional parameter, as it is not required in a two dimensional space. The default value is zero for it. Additionally, the object provides a `numpy` array for the coordinates directly as the property `n_vec`. There is also property called `__attached_elements__` which stores all attached element identifiers.

3.3 Finite Element Analysis

Solving the TO problem requires the performance a finite element analysis to calculate the displacements at the domain's nodes. For this, the analysis solves the elasticity problem of the domain, which is presented briefly in the next section. Furthermore, this subsection presents the finite element formulation used in this report including its implementation and highlights the global stiffness matrix. Lastly, the implementation for the FEA is outlined.

3.3.1 Elasticity Problem

The elasticity problem describes the connection between the displacements of a domain $\vec{u}$ and the systems load cases. In this report, the formulation

that is the easiest to solve used for this problem to keep the calculation time short.

$$\boldsymbol{S}\vec{u} = \vec{f}. \tag{38}$$

The proportional factor $\boldsymbol{S}$ describes the stiffness of the system and is also known as stiffness matrix. This matrix can be assembled from the individual finite elements stiffness matrices (3.3.4). The vector $\vec{f}$ maps the load cases in a global force vector and can be imagined as the forces which attack at the system's nodes. In the following, the element formulation is presented which is used in this report.

3.3.2 Element Formulation

In this thesis, only a simple linear element formulation for two dimensional rectangles is used. This formulation is sufficient to examine the feasibility of column generation for topology optimization problems. Each element is made of four nodes and only the translational degrees of freedom are considered, so that an element has eight degrees of freedom

$$\begin{aligned} e_n &= 4 \\ e_d &= 8. \end{aligned}$$

Furthermore, the elements are square with an edge length of one so that the surface or rather the volume of one element v_e is one. Due to these simplifications, it is much easier to assemble the domain's stiffness matrices because each element has the same stiffness matrix and it is independent from coordinates.

Shape Functions The shape functions for the four nodes are given by

$$\begin{aligned} N_1(\tilde{x}, \tilde{y}) &= \frac{1}{4}(1 - \tilde{x})(1 - \tilde{y}) \\ N_2(\tilde{x}, \tilde{y}) &= \frac{1}{4}(1 + \tilde{x})(1 - \tilde{y}) \\ N_3(\tilde{x}, \tilde{y}) &= \frac{1}{4}(1 + \tilde{x})(1 + \tilde{y}) \\ N_4(\tilde{x}, \tilde{y}) &= \frac{1}{4}(1 - \tilde{x})(1 + \tilde{y}) \end{aligned}$$

[Bat16] for the local coordinate system of an element. They can also be used to calculated element-specific forces.

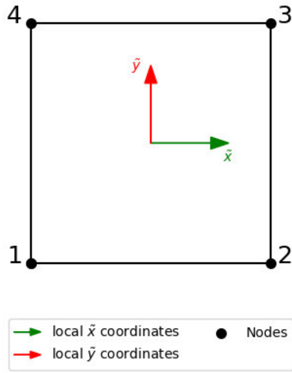


Figure 4: Visualization of used finite elements for topology optimization in this thesis

Stiffness Matrix A general element's stiffness matrix $\boldsymbol{S}_e$ can be defined by

$$\boldsymbol{S}_e := (s_{ij}) \in \mathbb{R}^{e_d \times e_d}.$$

Every stiffness matrix is square and symmetric, and without any boundary conditions a stiffness matrix is not invertible. For a detailed explanation of calculating an element's stiffness matrix the article "Introduction to the finite element method" by G.P. Nikishkov [Nik04] provides further information.

For this report, the element's matrix is given by

$$
\boldsymbol{S}_e := \frac{E}{1-\nu^2}
\begin{pmatrix}
\frac{1}{2}-\frac{\nu}{6} & \frac{1}{8}+\frac{\nu}{8} & -\frac{1}{4}-\frac{\nu}{12} & -\frac{1}{8}+\frac{3\nu}{8} & -\frac{1}{4}+\frac{\nu}{12} & -\frac{1}{8}-\frac{\nu}{8} & \frac{\nu}{6} & \frac{1}{8}-\frac{3\nu}{8} \\
\frac{1}{8}+\frac{\nu}{8} & \frac{1}{2}-\frac{\nu}{6} & \frac{1}{8}-\frac{3\nu}{8} & \frac{\nu}{6} & -\frac{1}{8}-\frac{\nu}{8} & -\frac{1}{4}+\frac{\nu}{12} & -\frac{1}{8}+\frac{3\nu}{8} & -\frac{1}{4}-\frac{\nu}{12} \\
-\frac{1}{4}-\frac{\nu}{12} & \frac{1}{8}-\frac{3\nu}{8} & \frac{1}{2}-\frac{\nu}{6} & -\frac{1}{8}-\frac{\nu}{8} & \frac{\nu}{6} & -\frac{1}{8}+\frac{3\nu}{8} & -\frac{1}{4}+\frac{\nu}{12} & \frac{1}{8}+\frac{\nu}{8} \\
-\frac{1}{8}+\frac{3\nu}{8} & \frac{\nu}{6} & -\frac{1}{8}-\frac{\nu}{8} & \frac{1}{2}-\frac{\nu}{6} & \frac{1}{8}-\frac{3\nu}{8} & -\frac{1}{4}-\frac{\nu}{12} & \frac{1}{8}+\frac{\nu}{8} & -\frac{1}{4}+\frac{\nu}{12} \\
-\frac{1}{4}+\frac{\nu}{12} & -\frac{1}{8}-\frac{\nu}{8} & \frac{\nu}{6} & \frac{1}{8}-\frac{3\nu}{8} & \frac{1}{2}-\frac{\nu}{6} & \frac{1}{8}+\frac{\nu}{8} & -\frac{1}{4}-\frac{\nu}{12} & -\frac{1}{8}+\frac{3\nu}{8} \\
-\frac{1}{8}-\frac{\nu}{8} & -\frac{1}{4}+\frac{\nu}{12} & -\frac{1}{8}+\frac{3\nu}{8} & -\frac{1}{4}-\frac{\nu}{12} & \frac{1}{8}+\frac{\nu}{8} & \frac{1}{2}-\frac{\nu}{6} & \frac{1}{8}-\frac{3\nu}{8} & \frac{\nu}{6} \\
\frac{\nu}{6} & -\frac{1}{8}+\frac{3\nu}{8} & -\frac{1}{4}+\frac{\nu}{12} & \frac{1}{8}+\frac{\nu}{8} & -\frac{1}{4}-\frac{\nu}{12} & \frac{1}{8}-\frac{3\nu}{8} & \frac{1}{2}-\frac{\nu}{6} & -\frac{1}{8}-\frac{\nu}{8} \\
\frac{1}{8}-\frac{3\nu}{8} & -\frac{1}{4}-\frac{\nu}{12} & \frac{1}{8}+\frac{\nu}{8} & -\frac{1}{4}+\frac{\nu}{12} & -\frac{1}{8}+\frac{3\nu}{8} & \frac{\nu}{6} & -\frac{1}{8}-\frac{\nu}{8} & \frac{1}{2}-\frac{\nu}{6}
\end{pmatrix}
\tag{39}
$$

[Sig] and it shows that the matrix only depends on the material properties, the Young's Modulus E and the Poisson's ratio ν.

Displacement Vector This formulation leads to the following general displacement vector for one element

$$\vec{u}_e = (u_{ed}) \in \mathbb{R}^8. \tag{40}$$

Density and Volume The Symbol ρ_e describes the density of a finite element and is more like a filling level. An element's basic volume v_e^0 is strongly simplified in this report and always equals one. The actual general volume of an element is

$$v_e := \rho_e \cdot v_e^0. \tag{41}$$

3.3.3 `Element` Object

The `Element` class represents the element formulation above. To create an instance of it, a dictionary with node identifiers and the corresponding node objects must be provided. In addition, the stiffness matrix (for a Young's Modulus E it equals one and for a Poisson's ratio ν it equals 0.3) of the element needs to be passed to the constructor. This is just an initial stiffness matrix for the optimization, as an update of it is required (cf. 4.6). The last parameter to create an element object is the element's density ρ_e which needs to be a value between zero and one. Element objects have a lot of helper functions and attributes which can be found in the explicit implementation in the repository. The most important attributes are:

- Stiffness Matrix `Ke`
- Density `x`
- Compliance `Ce`
- Derived Compliance `dCe`
- Centroid `centroid`

and the most important methods are:

- `update_stiffness_matrix`
- `set_centroid`
- `determine_neighbors`
- `update_compliance`
- `derive_compliance`

- `apply_sensitivity_filter`

Regarding the stiffness matrix and all further vectors or matrices, it is strongly suggested to implement the sparse form of them to increase the general performance of this approach. Furthermore, the implementation contains a prepared framework for arbitrary meshes, which depends on abstract classes.

3.3.4 Global Stiffness Matrix

The domain's stiffness matrix $\boldsymbol{S}$ is assembled from the individual elements' stiffness matrices $\boldsymbol{S}_e$ and can be defined as

$$\boldsymbol{S} := (s_{ij}) \in \mathbb{R}^{n_e \times n_e} \tag{42}$$

$$s_{ij} := \begin{cases} \boldsymbol{S}_i & : i = j \\ 0 & : i \neq j \end{cases} \tag{43}$$

while considering the elements. To receive the scalar form of the matrix, the degrees of freedom have to be considered so that the dimension equals the number of degrees of freedom of the complete domain. In this case, the stiffness matrix can be defined as sum of the element matrices

$$\boldsymbol{S} = \sum_{e \in M} \boldsymbol{S}_e.$$

This consideration is used for the implementation which is described in subsection 3.6.

3.4 Boundary Conditions

For solving the elasticity problem exactly, a domain also needs some boundary conditions to make the stiffness matrix invertible, so that $\boldsymbol{S}^{-1}$ exists. There are two types of conditions: fixations and load cases.

3.4.1 Fixations

Physically, the geometry is held in position by fixations. Mechanically, they can determine a system statically which defines the way the FEA is solved. Mathematically, fixations are boundary values for the elasticity problem which is solved for the FEA. Here, fixations are applied to nodes of a domain as the corresponding displacements are zero. The set N_f describes the global node indices, which are fixed in space, and the set D_f describes the corresponding DoFs.

3.4.2 Load Cases

Load cases are, like fixations, user-defined boundary conditions. All load cases define the force vector

$$\vec{f} := (f_i) \in \mathbb{R}^{n_d} \tag{44}$$

for the elasticity problem (4) of the domain. In this report, forces are also applied to nodes directly.

3.4.3 `BoundaryConditions` Object

The current implementation contains an separate class to consider boundary conditions of a domain. To avoid providing the complete outer force vector $\vec{f}$, as it can be very large, the constructor of the `BoundaryConditions` class takes a dictionary to map them. A key of the dictionary describes the node where a force attacks and the value is the corresponding force vector at the node. Below, an example can be found:

```
import numpy as np
f = {2: np.array([0, -1, 0])}
```

This example defines a force gripping at node two and directs in negative y-direction. Using dictionaries has the advantage of defining multiple load cases in individual dictionaries and merging them into one resulting dictionary. An example for multiple load cases is provided below:

```
import numpy as np
f1 = {2: np.array([0, -1, 0])}  # concentrated force at
    node 2

q1 = {
        3: np.array([0, 1, 0]),
        11: np.array([0, 1, 0])
        12: np.array([0, 1, 0])
}  # line load for nodes 3, 11 and 12

F = dict(f1.items() + q1.items())  # import dict for BC
```

Currently, it is just possible to define totally fixed nodes in space by providing a list of fixed nodes as shown in the example below

```
fixed_node = [1, 5]
```

This list holds the nodes one and five in their position. A dictionary to define the outer forces, a list of fixed nodes, a list of all node identifiers and the number of degree of freedoms per node are required to create an instance of `BoundaryConditions`.

3.5 Solving Elasticity Problems

While performing a FEA and solving the elasticity problem, two main steps are carried out:

1. Incorporating boundary conditions
2. Solving reduced linear equations system (LES) depending on static determinacy

During the first step, the LES of the elasticity problem is reduced by incorporating the fixation boundary condition. After that, the reduced LES is solved by a direct or an iterative solver, depending on the static determinacy.

3.5.1 Incorporating Boundary Conditions

The fixed nodes change the original elasticity problem to a reduced one. Mechanically, they remove the degrees of freedom of the corresponding nodes, so that the nodes can not move along that DoFs anymore. If the configuration of boundary conditions does not allow any rigid body motions for the geometry, the model is statically determined and a direct solver can be used. The reduced global stiffness matrix is calculated by

$$\hat{\boldsymbol{S}} = (\hat{s}_{i,j}) \in \mathbb{R}^{n_d \times n_d} \tag{45}$$

$$\hat{s}_{i,j} = \begin{cases} s_{i,j} & : i,j \notin D_f \\ 0 & : i,j \in D_f \wedge i \neq j \\ 1 & : i,j \in D_f \wedge i = j \end{cases} \tag{46}$$

and the reduced force vector is defined by

$$\vec{f}_{red} = (\hat{f}_i) \in \mathbb{R}^{n_d} \tag{47}$$

$$\hat{f}_i = \begin{cases} f_i & : i \notin D_f \\ 0 & : i \in D_f \end{cases}. \tag{48}$$

For choosing the solver for the reduced LES, there are at least two properties which lead to the direct exact solving method:

- $rank(\hat{\boldsymbol{S}}) = n_d$
- $det(\hat{\boldsymbol{S}}) \neq 0$

With these properties, the LES will have one exact solution.

3.5.2 Direct

As long as the finite element system is statically determined, the resulting linear equations system of the elasticity problem can be solved directly and an exact solution can be found. Static determinacy means the existing of the inverse stiffness matrix $\hat{\boldsymbol{S}}^{-1}$, so that

$$\vec{u} = \hat{\boldsymbol{S}}^{-1}\vec{f} \tag{49}$$

and the displacement vector $\vec{u}$ can be calculated directly, which is also exact. The implementation uses the function `linalg.solve(a,b)` from the `NumPy` package [Num], which solves a LES like

$$\boldsymbol{A}\vec{x} = \vec{b}. \tag{50}$$

In this case, the input parameter `a` is the reduced stiffness matrix $\hat{\boldsymbol{S}}$ and the parameter `b` corresponds to the reduced force vector $\vec{f}_{red}$. This function will return the displacements $\vec{u}$ regarding a stiffness matrix and a force vector.

3.5.3 Iterative

As this report also considers domain decomposition for TO problems (3.8), it is required to solve static undetermined finite element systems. Static undeterminacy means that it unclear whether the inverse of a stiffness matrix exists so rigid body motions of the domain are possible. There are not enough boundary conditions which can be incorporated to the stiffness matrix. The resulting LES has an infinite number of solutions but most of these solutions are not reasonable for the problem. One constraint for a main domain is the static determinacy, so that an exact solution can always be calculated for it and an initial solution vector

$$\vec{u}_{init} := (u_{init,i}) \in \mathbb{R}^{n_d} \tag{51}$$

can be provided, which is used to define a trust region or rather bounds for the displacement vector $\vec{u}$. The displacements change during the optimization but it is not expected that the direction of a displacement changes because the loads are constant. Regarding the range of the trust region, a factor ν

estimates the bounds.

$$
\begin{aligned}
\vec{u}^{+} &:= (u_i^{+}) \in \mathbb{R}^{n_d} \\
u_i^{+} &:= \max(\{u_{0,i} \cdot \nu,\ \ u_{0,i} \cdot (1-\nu)\}) \\
\vec{u}^{-} &:= (u_i^{-}) \in \mathbb{R}^{n_d} \\
u_i^{-} &:= \min(\{u_{0,i} \cdot \nu,\ \ u_{0,i} \cdot (1-\nu)\}) \\
\nu &\in [0,1]\,.
\end{aligned}
$$

If the estimation factor ν is too small, no solution will be found for this problem. But if the factor is too large, there will be an unreasonable solution for the problems. It may be possible to find the optimal value for ν by considering a maximum allowed stress, but it is still possible to find a good solution by guessing a factor. A reasonable solution for the static undetermined domain can be estimated by defining a least-squares problem and considering the trust region for the displacements in this problem.

$$
\mathit{min} \qquad ||\hat{\boldsymbol{S}}\vec{u} - \vec{f}_{red}|| \tag{52}
$$

$$
s.t. \qquad \vec{u}^{-} \leq \vec{u} \leq \vec{u}^{+}. \tag{53}
$$

This problem formulation is used to receive a reasonable solution for a static undetermined domain. It is necessary to keep in mind that this only works well if an initial solution for a similar domain is known. In addition, it needs to be investigated how this approach for solving a static undetermined system impacts the global optimality of the CG approach. As there are initial values for the displacement, it is possible to determine a reasonable trust region for the problem, so the global optimality should not be restricted. However, there is no evidence for this.
The implementation uses the function `optimize.least_squares(...)` from the `SciPy` module [Sci]. Detailed information on the explicit implementation is provided in section 3.6.3.

3.6 `FEModel` Object

This section presents the implementation view of a finite element analysis for a topology optimization domain. The implemented class `FEModel` provides all properties and methods that are required to perform a FEA for a domain. This class can be applied to each domain, regardless of whether it is a main domain or sub-domain. The class can be found in ”pylib/fea/fea.py”.

3.6.1 Constructor

The constructor takes a `list` of node identifiers of the domain, an element `dictionary` from the corresponding domain and the applied boundary conditions as a `BoundaryConditions` object as required input parameters. Besides creating an instance of this object, the constructor examines the static determinacy of the domain by

```
if len(self.boundary_conditions.fixed_nodes) > 1:
        self.is_undetermined = False
else:
        self.is_undetermined = True
```

As long as the implementation provides just holding complete nodes in space and single degrees of freedom, this way of testing is sufficient.

3.6.2 Attributes

This class provides several attributes regarding a FEA and the most important ones can be found in the table below.

Attribute	Type	Short Description
`F`	`numpy.array`	Force Vector $\vec{f}$; points to boundary conditions force vector
`K`	`numpy.array`	Stiffness Matrix $\boldsymbol{S}$
`U`	`numpy.array`	Displacement Vector $\vec{u}$
`R`	`numpy.array`	Reaction Forces at fixed nodes
`boundary_conditions`	`BoundaryConditions`	Boundary Conditions object
`is_undetermined`	`bool`	Static determinacy; leads in used solver for FEA
`u_base`	`numpy.array`	Represent the initial displacement vector $\vec{u}_{init}$
`u_bounds`	`tuple(numpy.array, numpy.array)`	Stores bounds for trust region approach

Table 2: Most Important Attributes of FEModel class

Regarding the attribute `u_bounds`, the first entry of the `tuple` are the lower bounds $\vec{u}^-$ and the second entry maps the upper bounds $\vec{u}^+$. Detailed information can be found directly in the implementation.

3.6.3 Methods

The class for the finite element analysis contains some very important methods that are absolutely necessary for the topology optimization problems. The following paragraphs in this section summarize briefly the most important parts of them.

Function `get_k` This method assembles the domain's stiffness matrix and returns it. There are two ways to apply this method. If there is no density

distribution provided for the optional input parameter _x, the densities of the element objects or rather the element stiffness matrices will be used directly for it. If the user provides a density distribution in accordance to the domain, this method will use this density distribution to assemble the domain's stiffness matrix by using the SIMP approach (4.1). The method applies an optional parameter, which is described later (4.1.1). The specific description for the method is shown below:

```
def get_k(self, _x=None, p=3):
        k = np.zeros((self.dof_number, self.dof_number))
        for dict_it, (e_id, element) in enumerate(self.elements.items()):
                local_dofs = np.where([self.global_dofs == d for d in
                    element.DoFHelper])[1]
                for it, i in enumerate(local_dofs):
                        for jt, j in enumerate(local_dofs):
                                if _x is None:
                                        k[i, j] += element.Ke[it, jt]
                                else:
                                        k_val = ((1e-9 + _x[dict_it] ** p *
                                            (1 - 1e-9)) * element.Ke0)[it,
                                            jt]
                                        k[i, j] += k_val
        return k
```

It is highly recommended to rewrite this method using a sparse form to improve the performance. The non-sparse form is used because the models described in this report are simple and this method provides a clearer view regarding assembling a domain's stiffness matrix.

Function `incorporate_boundary_conditions` The functionality to incorporate the fixed nodes boundary condition into a stiffness matrix and a force vector is provided by the method `incorporate_boundary_conditions`. The stiffness matrix and the force vector are required input parameters. The function returns the reduced stiffness matrix and the reduced force vector according to section 3.5.1. Below, the specific implementation is outlined:

```
def incorporate_boundary_conditions(self, k: np.array, f: np.array):
        local_fixed_dofs = np.where([self.global_dofs == d for d in
            self.boundary_conditions.fixed_dofs])[1]
        for i, d in enumerate(local_fixed_dofs):
        k[:, d] = 0
        k[d, :] = 0
        k[d, d] = 1
        f[d] = 0
        return k, f
```

Applying the sparse form may change this method as well.

Function `set_bounds` To define the bounds of the displacements for static undetermined domain, the method `set_bounds` is implemented. Regarding the function to solve the least-squares problem, it is required that every

lower bound must be truly lower than the corresponding upper bound. This leads to a brief adjustment in the implementation as this constraint is not ensured if an initial displacement is zero. Below, the implemented function is presented:

```
def set_bounds(self, u_domain):
        self.u_base = u_domain
        _ub = np.array(((1 - nu) * u_domain, nu * u_domain)).T
        for i in range(len(_ub)):
                _ub[i] = _ub[i, np.argsort(_ub[i])]
        self.u_bounds = (
        _ub[:, 0],
        _ub[:, 1]
        )
        self.u_bounds[0][self.u_bounds[0] == 0] -= 1e-3
        self.u_bounds[1][self.u_bounds[1] == 0] += 1e-3
```

Regarding the last two lines, it is necessary to ensure that each lower bound is truly lower than the corresponding upper bound. This method changes the class attributes instead of returning the new bounds.

Function `solve` This method performs the FEA and solves the elasticity problem for the corresponding domain described in the object. It is also possible to provide a user-specific reduced force vector to the input parameter `f_user`. If this parameter is `None`, this method will use the object's force vector. The method contains two different solvers, one for a static determined system and one for an undermined system. Which solver is used makes no difference to the return as it is always the domain's displacements and a boolean value which describes whether a solution is feasible or not in terms of the displacement bounds. Calling this method will always update the domains attributes for the displacements and reaction forces. Furthermore, this method updates the element objects of the domain. This update includes the elements' nodal displacements and nodal forces.
Using the direct solver for a static determined system is quite simple as the corresponding `NumPy` function, `numpy.linalg.solve`, can be called directly by passing the domain's stiffness matrix and the required force vector to it. The solution will always be feasible. To use the iterative solver for the least squares approach, it is required to define a function which represents the objective. In the following, the implemented part for the iterative approach is shown.

```
        # make the objective function for least-squares problem
        def make_func(_k, _f):
                def func(_u):
                        return np.dot(_k, _u) - _f
                return func
        # solve the indeterminate system using scipy's least-squares solver
        res = least_squares(make_func(self.K, _f), self.u_base,
            bounds=self.u_bounds, method='dogbox')
```

```
        self.U = res.x
```

To map the objective, an enclosure for the required function is used. It returns the function object which can be used for the `SciPy` function `optimize.least_sqaures`. This is required because the stiffness matrix changes during the optimization. The parameter `u_base` defines the starting point for the algorithm and equals the initial displacements of the FEA for the original domain. The method parameter defines the method or rather the algorithm which is used to solve the problem. After testing the different possible methods, the "dogleg"-algorithm generated the best results in terms of time, quality and stability. Further information can be found here: `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html` [Sci].

3.7 Experimental Model

The experimental model contains 25 nodes and 16 elements in a two-dimensional design space. In accordance to the element formulation from section 3.3.2, this leads to a total degree of freedom number of 50. The shape is square with four elements each in x- and y-direction. Regarding the boundary conditions, there are two fixed nodes and one force attacks on the lower right node with a value of one in negative y-direction. The main-domain is shown in the figure below.

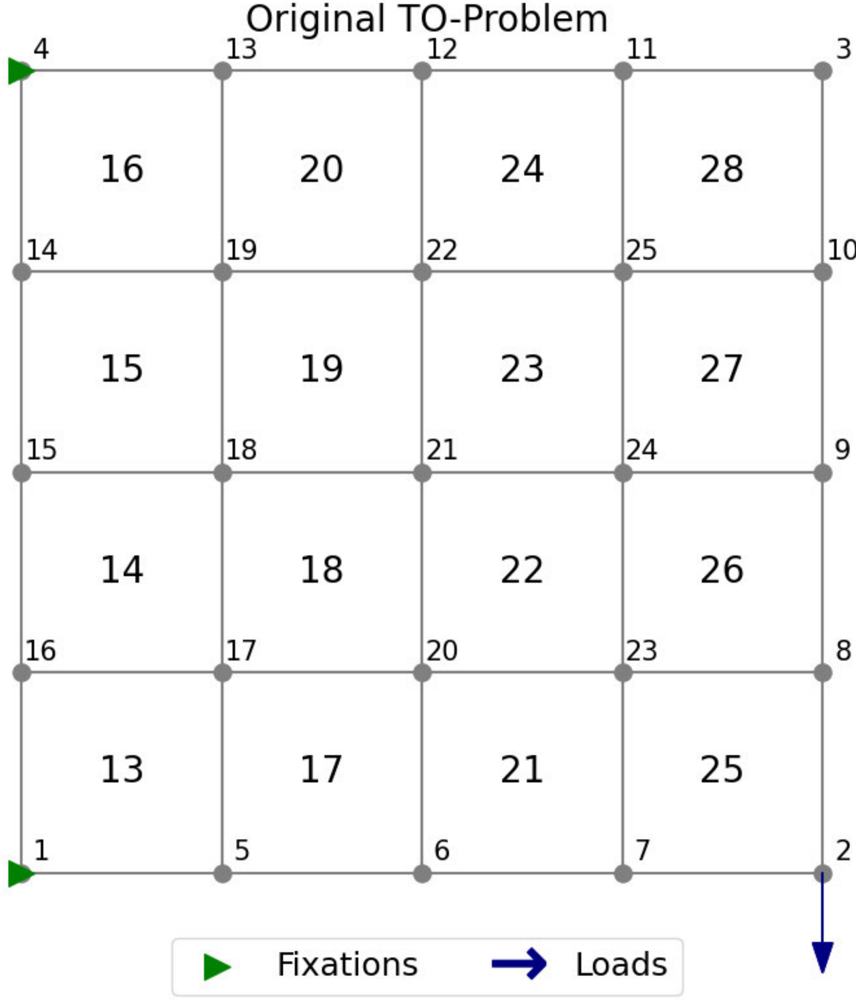


Figure 5: Experimental Model

Properties for the TO problem description can be found in the table below.

Name	Symbol	Value
Geometry		
Number of Nodes	n_n	25
Number of DoFs	n_d	50
Number of Element	n_e	16
Boundary Conditions		
Fixed Nodes	N_f	$\{1,4\}$
Loaded Nodes	N_l	$\{2\}$

The force vector at node two equals

$$\vec{f_2} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}$$

and is fixed during the optimization process. Due to the boundary conditions, the main-domain is statically determined and a direct solver for the FEA can be used. This is a constraint for TO models in this report. Since this thesis only determines the prove of concept of the new approach, the model was deliberately chosen to be so simple in order to go out of scope and to make it as easy as possible to obtain initial results.

3.7.1 Mapping Node to Degree of Freedom

Often, it is required to calculate the indices of degrees of freedom to a corresponding node and vice versa. The following functions describe the connection between node numbers and numbers of degree of freedom. The functions depend on the element formulation and in particular on the number of degrees of freedom per node n_{d_n}. Indices start counting from zero

$$d(n) = \begin{pmatrix} 2 \cdot n \\ 2 \cdot n + 1 \end{pmatrix}$$

$$n(d) = \begin{cases} \frac{d}{n} & : d \bmod n = 0 \\ \frac{d-1}{n} & : d \bmod n = 1 \end{cases}.$$

These two functions only work if the number of degree of freedom at a node is two. It is possible to write them more generally, but it is not necessary here. The function $d(n)$ returns the indices of degrees of freedom for a given node index n while the other function $n(d)$ provides the node index n to a given degree of freedom index d.

3.7.2 Global Force Vector

The load case only contains one force at node two with the index one. Hence, the corresponding DoF indices are

$$d(1) = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

taking into consideration that there is only a force in y-direction. The global force vector $\vec{f}$ for the main-domain is defined by

$$\vec{f} := (f_d) \in \mathbb{R}^{n_d}$$
$$f_d = \begin{cases} -1 & : d = 3 \\ 0 & : d \neq 3 \end{cases}.$$

This vector is constant during the optimization process.

3.8 Domain Decomposition of TO-Problem

The blocks k of the reformulated topology optimization problem, the block TO problem (2.2), can be understood as sub-domains Ω_k of the main-domain Ω and define a TO problem itself. This section shows the decomposition which can be applied to a main-domain and is currently implemented in the layer. As this report determines just the feasibility whether it makes sense to use column generation for topology optimization problems, it is not fully clear if the shown decomposition performs best. There are some suggestions for further investigations for the domain decomposition. This chapter describes the decomposition by means of the experimental model, which is also used for some first investigations using column generation for a TO problem. The experimental model corresponds to the main domain Ω and will be cut by two lines in four sub-domains. The cut lines applied to the main domain are shown in figure 6.

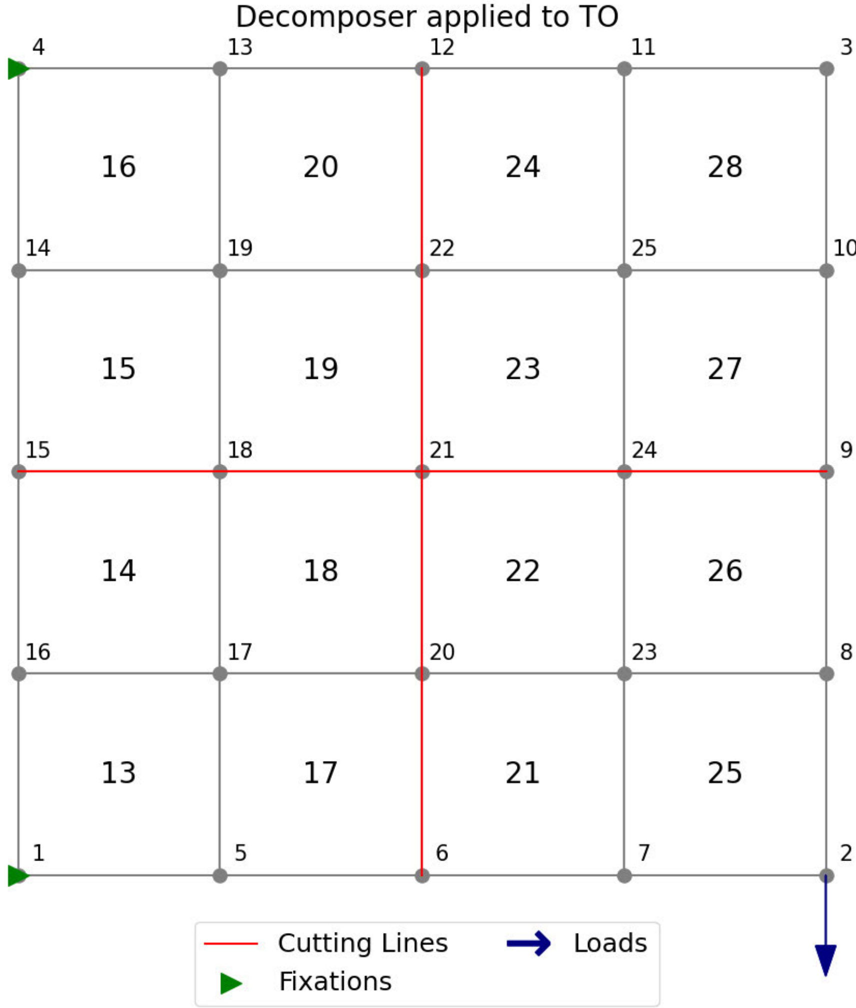


Figure 6: Cutting Lines for Sample Main-Domain

3.8.1 Sub-Domain Definition

There are several options for domain decomposition of finite element systems as it is a separate field of research. It is divided into two areas: overlapping and non-overlapping methods. In this report, a simple non-overlapping method is used to decompose the main domain. In addition, it must be noted that this method is targeted specifically at solving topology optimization problems, as a FEA is executed again and again during the process. The decomposition is based on an initial FEA for the complete model, so it becomes clear that this method is not suitable for a classic FEA. By cutting

nodes of the main-domain, the sub-domains are created. A sub-domain Ω_k is very similar to the main-domain Ω and should define a TO problem itself in terms of the required sub-problems for the CG algorithm. The following table highlights some basic properties of a sub-domain that are very similar to the properties of a main domain (table 1). In addition, their connections to the main-domain are highlighted. In general, an index k denotes a sub-domain symbol, whereas the main symbol stays the same.

Name	Symbol	Connection to Main-Domain
Set of Elements	M_k	$M_k \subseteq M$
Set of Nodes	N_k	$N_k \subseteq N$
Set of DoFs	D_k	$D_K \subseteq D$
Volume	V_k	$V = \sum_{k \in K} V_k$

Table 3: Basic Properties of a sub-domain Ω_k

In addition, a sub-domain needs a definition of the boundary to its neighboring sub-domains. Γ_k denotes this boundary of a sub-domain Ω_k. This definition is required to map the interaction between the sub-domains among each other. Furthermore, a sub-domain does not need to be statically dertermined, as the iterative FEA can be applied to the sub-domain (3.5).

3.8.2 Sub-Domain's FEA

As a sub-domain Ω_k should define sub-problems, it must be possible performing a FEA for it by solving the elasticity problem for the sub-domain

$$\boldsymbol{S}_k \vec{u}_k = \vec{f}_k. \tag{54}$$

The stiffness matrix of a sub-domain is defined as

$$\boldsymbol{S}_k := \sum_{e \in M_k} \boldsymbol{S}_{ke}. \tag{55}$$

and corresponds to the definition of the stiffness matrix of Ω with the difference that just the elements of Ω_k are used for the required domain's stiffness matrices $\boldsymbol{S}_k$. The domain's displacement vector is defined by

$$\vec{u}_k := (u_{ki}) \in \mathbb{R}^{|D_k|} \tag{56}$$

and the force vector of the sub-domain is

$$\vec{f_k} := (f_{ki}) \in \mathbb{R}^{|D_k|}. \tag{57}$$

This is very similar to the FEA of the main-domain and corresponds to the reformulation of a TO problem.

3.8.3 `SubDomain` Object

This section outlines the most important aspects of the `SubDomain` class which maps the sub-domain definition, including the required methods for performing the FEA and solving the TO problem. The class is placed in file "pylib/tomodel/subdomain.py" Each `SubDomain` object is assigned an unique integer identifier `SubDomain_ID`. Like done for the `TOModelBase` object, an instance of this class includes a `dict` of nodes and a `dict` of Elements regarding the sub-domain. Furthermore, each object of the `SubDomain` class provides properties to represent the related FEA and SIMP solver.
Besides some simple functions, such as determining the current volume or receiving the current density distribution, the class provides two very significant methods regarding the functionality. Calling the method `update_domain` sets all important attributes, including properties for the FEA and SIMP solver. Furthermore, this method sets the boundary conditions of the sub-domain so it needs to be called after instantiating an object of `SubDomain`. The other significant function is `get_boundary_forces`, which is described in the following chapters. More detailed information can be found in the explicit implementation in the repository.

3.8.4 Boundary Definition

Each sub-domain Ω_k has a boundary Γ_k to its neighboring sub-domains. As the domain decomposition cuts nodes and not elements, a boundary contains boundary nodes Γ_{kn} which lead to the corresponding boundary DoFs Γ_{kd} Both are sub-sets of the relating domain's sets

$$\begin{aligned} \Gamma_{kn} &\subseteq N_k \\ \Gamma_{kd} &\subseteq D_k. \end{aligned}$$

.

3.8.5 Boundary Forces

Just cutting the main-domain Ω in sub-domains Ω_k can lead to very ill-posed sub-problems such as

- insufficient number of boundary conditions from original problem for statically determinacy
- sub-problem does not contain any outer forces $\vec{f}_k = \vec{0}$

The first issue is solved by using the iterative solver (3.5.3) for the elasticity problem. The required boundaries for the displacements $\vec{u}^+$ and $\vec{u}^-$ have been calculated in an initial FEA of the main-domain. This is has been done because it is not expected that displacements change its direction during the TO, as the load cases are constant. The factor ν can be estimated as the displacements must not extremely large because that would increase the compliance which is going to be minimized. The bounds describe a trust-region for the least-squares problem.
Solving the elasticity problem of the main-domain leads to the observation that in each relating sub-domain, displacements occur even if it does not contain an outer force from the original problem. To remove this uncertainty, boundary forces $\vec{g}_k$ are introduced to the sub-domain's elasticity problem

$$\boldsymbol{S}_k \vec{u}_k = \vec{f}_k + \vec{g}_k. \tag{58}$$

The basic idea for the boundary forces was found in the paper "Solution of the Topology Optimization Problem Based Subdomains Method" [ARE08] but the idea is modified in this report, as the article does not consider ill-posed sub-problems. The method from the article determines the stresses at a boundary, whereas the domain decomposition from this report calculates the forces at a sub-domain's boundary to modify the right hand side of the elasticity problem in accordance to initial displacements. They grip on the boundary nodes as the right-hand side considers outer gripping forces of a finite element system. Figure 7 outlines the idea of the boundary forces by meaning of the experimental model.

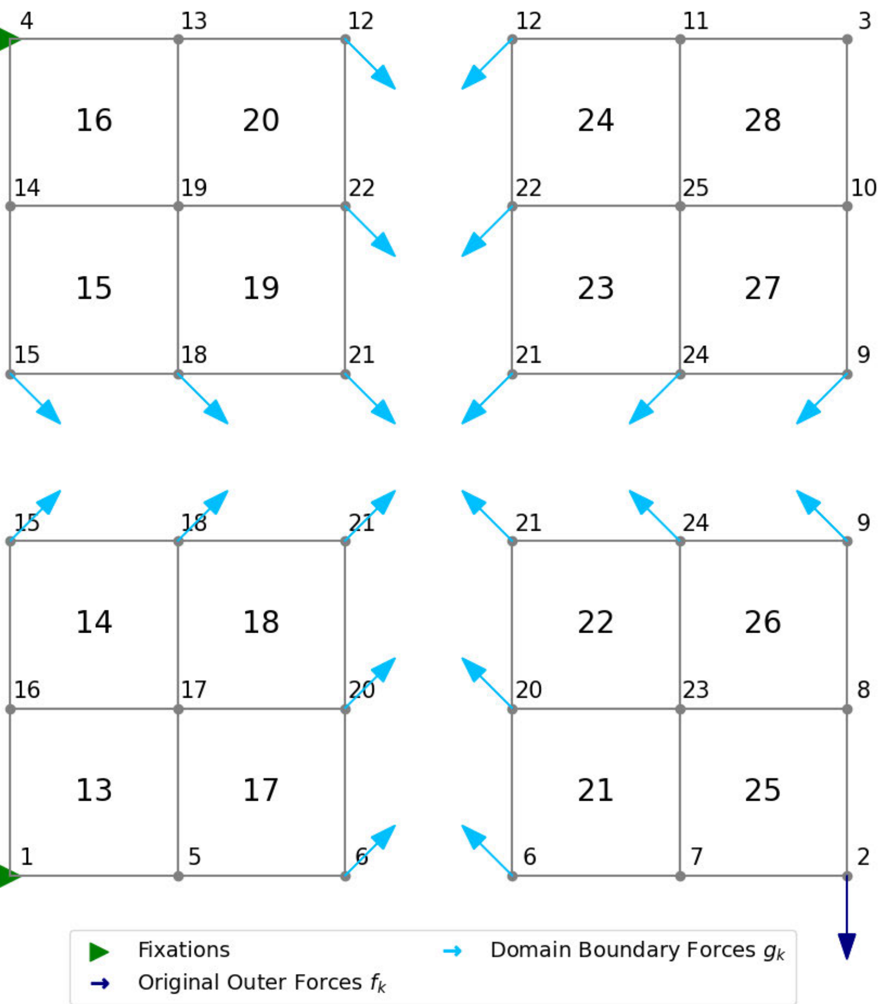


Figure 7: Idea of Boundary Forces

The light-blue arrows of the boundary forces are not scaled and directed according to the true load case, as they just outline the idea of the boundary forces.
There are two methods to calculate the boundary forces $\vec{g_k}$. The easier method is evaluating the elasticity equation (equation 55) in terms of the initial displacements $\vec{u}_{k,init}$

$$\vec{g}_k = \boldsymbol{S}_k \vec{u}_{k,init} - \vec{f}_k. \tag{59}$$

as it directly provides the required right-hand side of the elasticity problem, which can be used for the sub-domain's TO problem. This method is mostly

considered in this report.
There is another way to calculated those boundary forces using the Gaussian integration points of the element formulation, but there are ongoing research on it. One expected advantage of this method is the calculating of every elements' forces in one single method and uncovering connections between them.

Function `get_boundary_forces` The function `get_boundary_forces` calculates the boundary forces $\vec{g}_k$ depending on given displacements and a density distribution. The density distribution is required to calculate the stiffness matrix $\boldsymbol{S}_k$. As described in the section above, two methods are implemented to calculate the boundary forces whereby the selection is steered by the optional string parameter `method`. The value of this parameter can be 'ku' or 'gauss'. In addition, a boolean value can be passed to the function to update the corresponding class attributes, while calculating the forces. A dictionary, which has the same structure as the dictionary for the outer forces, and a `numpy` array is returned for the boundary forces. Below, the current implementation for the simple 'ku' method is provided.

```
g_dict = {}  # initialize g_dict; not required for g_vec
if method == 'ku':
        # solve k_domain dot u_domain - f_domain_outer
        # get k for corresponding density distribution
        _k = self.FEModel.get_k(_x=x)
        if len(self.FEModel.boundary_conditions.fixed_nodes) > 0:
                _k, _ =
                    self.FEModel.incorporate_boundary_conditions(_k,
                    self.FEModel.F)
        g_vec = np.dot(_k, u) -
            self.FEModel.boundary_conditions.outer_forces_vec
        # create the return dict
        for boundary_node in self.domain_boundary_nodes:  # there
            is an entry in dict for every boundary node
                ndofs = get_dofs(boundary_node)  # get the global
                    dofs for the node
                # extract the force vector from g_vec
                _g = g_vec[np.array([self.get_local_dof_index(d)
                    for d in ndofs])[:] - 1]
                # set entry
                g_dict[boundary_node] = _g
```

The complete implementation including the second 'gauss' method can be found in the `SubDomain` class.

3.8.6 Incorporate Boundary Forces to Objective Function

In accordance of the reformulated block TO problem, the usage of the boundary forces must not change the original and reformulated TO problem. It needs to be proven that the incorporation of the boundary forces for the

blocks by cutting the main domain does not change the original problem. The objective function of the block-separable TO problem shows that the sum of the single compliances of each block equals the total compliance and the following condition for the boundary forces and boundary displacements can be derived

$$\begin{aligned}
c &= \sum_{k \in K} c_k \\
\iff \sum_{k \in K} \vec{u}_k^T \cdot \vec{f}_k &= \sum_{k \in K} \vec{u}_k^T \cdot (\vec{f}_k + \vec{g}_k) \\
\iff \sum_{k \in K} \vec{u}_k^T \cdot \vec{f}_k &= \sum_{k \in K} \vec{u}_k^T \cdot \vec{f}_k + \vec{u}_k^T \cdot \vec{g}_k \\
\iff \sum_{k \in K} \vec{u}_k^T \cdot \vec{f}_k &= \sum_{k \in K} \vec{u}_k^T \cdot \vec{f}_k + \sum_{k \in K} \vec{u}_k^T \cdot \vec{g}_k \\
\iff \sum_{k \in K} \vec{u}_k^T \cdot \vec{g}_k &= 0.
\end{aligned}$$

As the boundary forces just occur in a boundary, the boundary forces cause displacements at the corresponding boundary nodes so that

$$u_{ki} \cdot g_{ki} \neq 0, \;\; k \in K, \;\; i \in \Gamma_{k,d}, \tag{60}$$

but it is also possible to prove that the boundary forces cancel each other out, hence

$$\vec{g} = \sum_{k \in K} \vec{g}_k = \vec{0} \tag{61}$$

by means of a simple one-dimensional example. One important and obvious condition is the equality of displacements at the artificial added nodes (copy constraints) on a block's boundary Γ_k, so copied nodes are subject to the same displacements as the original node

$$u_{ki} = u_{lj}, \;\; k, l \in K, \;\; i \in \Gamma_{k,d}, \;\; j \in \Gamma_{l,d}. \tag{62}$$

A simple example model is given by

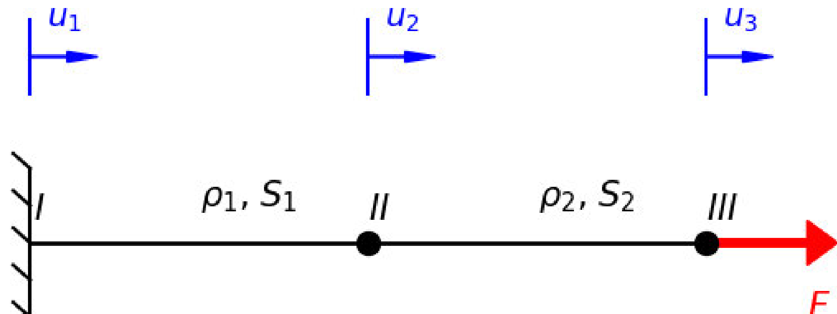


Figure 8: Model to prove $\vec{g} = \vec{0}$

with the corresponding compliance

$$c = u_3 \cdot F. \tag{63}$$

The next figure shows a corresponding block model in which the original model is decomposed in two sub-domains and node I is copied

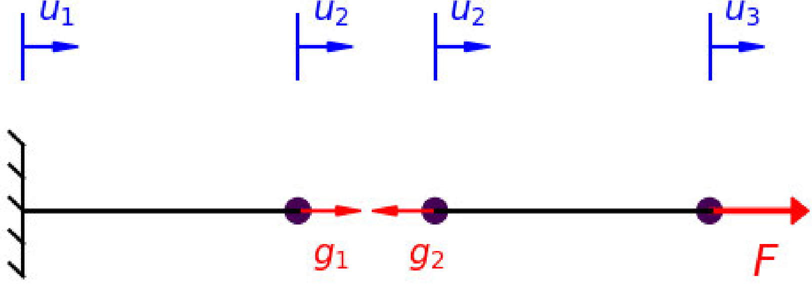


Figure 9: Block Model

The compliance of the block model is given by the sum of the single compliances of each block

$$c = c_1 + c_2 = g_1 \cdot u_1 + g_2 \cdot u_1 + F \cdot u_2. \tag{64}$$

As the total compliance of the original problem must remain unchanged, this equation leads to the following condition for the boundary forces

$$g_1 = -g_2. \tag{65}$$

The domain decomposition or rather the block creation depends on initial displacements

$$\tilde{u} = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \tag{66}$$

which are calculated using the elasticity problem as follows

$$\begin{pmatrix} S_1 & -S_1 & 0 \\ -S_1 & S_1 + S_2 & -S_2 \\ 0 & -S_2 & S_2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} F_L \\ 0 \\ F \end{pmatrix},$$

where F_L denotes to the reaction force at the fixation. Hence, including the boundary condition for the fixation on the left side

$$\begin{aligned} u_1 &= 0 \\ u_2 &= \frac{F}{S_1} \\ u_3 &= \frac{F(S_1 + S_2)}{S_1 \cdot S_2}. \end{aligned}$$

Solving the elasticity problems for the blocks leads to the boundary forces g_1 and g_2

$$\begin{pmatrix} S_1 & -S_1 \\ -S_1 & S_1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} F_L \\ g_1 \end{pmatrix}$$

$$\rightarrow g_1 = S_1 \cdot u_2$$

$$\begin{pmatrix} S_2 & -S_2 \\ -S_2 & S_2 \end{pmatrix} \begin{pmatrix} u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} g_2 \\ F \end{pmatrix}$$

$$\rightarrow g_2 = S_2 \cdot u_2 - S_2 \cdot u_3.$$

Hence,

$$
\begin{aligned}
g_1 &= -g_2 \\
S_1 \cdot u_2 &= -(S_2 \cdot u_2 - S_2 \cdot u_3) \\
S_1 \cdot \frac{F}{S_1} &= -S_2 \cdot \frac{F}{S_1} + S_2 \cdot \frac{F(S_1 + S_2)}{S_1 \cdot S_2} \\
F &= -\frac{F \cdot S_2^2}{S_1 \cdot S_2} + \frac{F \cdot S_2 \cdot (S_1 + S_2)}{S_1 \cdot S_2} \\
F &= F\frac{S_2(-S_2 + S_1 + S_2)}{S_1 \cdot S_2} \\
F &= F\frac{S_1 \cdot S_2}{S_1 \cdot S_2} \\
F &= F
\end{aligned}
$$

The proof above shows that the incorporation of the defined boundary conditions, based on the initial displacements, does not change the original problem and can be used in the reformulation to reach well-posed sub-problems. This also clearly defines the objective for the reformulated topology optimization problem

$$min \sum_{k \in K} \vec{u}_k^T \cdot (\vec{f}_k + \vec{g}_k). \tag{67}$$

Regarding the summarized block variable $\underline{x}_k$, the objective function is given by

$$min \sum_{k \in K} \underline{c}_k^T \cdot \underline{x}_k \tag{68}$$

$$\underline{c}_k := \big((0) \in \mathbb{Z}^{n_{ke}}, \vec{f}_k\big)^T \in \mathbb{R}^{n_k}. \tag{69}$$

3.8.7 `TODecomposer` Object

An object of the class `TODecomposer` can be used to perform the presented domain decomposition. It will decompose and reformulate an `TOModelBase` object so that it is prepared for Decogo. For instantiating, a list of the cut nodes numbers a dictionary of sub-domains and the base model needs to be passed to the constructor. To improve the process, it should be possible

to develop an algorithm to decompose a main-domain automatically, just by providing a list of cut node numbers. The dictionary of sub-domains contains the number of the sub-domain as the key and a "raw" object of `SubDomain` as the corresponding value. The constructor of the `TODecomposer` class updates the sub-domains, by inserting the corresponding boundary conditions and setting the initial displacements from the main-domain. This class provides some mappings for nodes, DoFs and sub-domains. To apply the decomposer, the class `TOModelBase` provides the function `reformulate` which can be called by passing an instance of the decomposer class to it. The current implementation is not optimized, but it is sufficient for testing the new approach.

4 Approaches for Solving a TO-Problem

The approaches presented in this report vary the Young's Modulus E to course a geometry change. Each element can be assigned a different Young's Modulus and this will be updated during the process. The updated Young's Modulus is used in an iteration step. If an element has a low density, its compliance will be high because the corresponding displacements will be high. Due to the low compliance, the corresponding element with the low Young's Modulus does not contribute to the strength of the geometry and can be omitted. This has led to the idea of connecting the Young's Modulus with the density of an element.
Topology optimization problems should result in a solution, where the densities of the elements are zero or one so an element is switched on or off. Mathematically, the problem would be a MINLP problem. As the number of elements and nodes can be very large and the problem is non-convex, TO problems can be very difficult to solve. The approaches used in this report convert the MINLP problems to NLP problems by connecting the Young's Modulus and the density of an element continuously. This is a strong simplification and leads to the heuristic character of the conventional approaches. Unless otherwise stated, the general descriptions of the different approaches base on the theory user manual of the freeware topology optimization software "Z88Arion" [[Z88]], which was published by the University Bayreuth.

4.1 OC - Optimality Criterion

The optimality criterion approach is used while minimizing the compliance c of a finite element system. For this approach, the Lagrangian equation for the general TO problem is formed,

$$\mathcal{L}(\underline{\rho}, \eta, \underline{\nu}, \underline{\mu}) = \vec{u}^T \boldsymbol{S} \vec{u} + \nu(V(\underline{\rho}) - V_f) + \sum_{e \in M} (\lambda_e(\rho_e - \rho^{max}) + \mu_e(\rho^{min} - \rho_e)) \quad (70)$$

and the optimal criteria

$$\nabla L(\underline{\rho}, \eta, \underline{\nu}, \underline{\mu}) = 0 \quad (71)$$

can be set up. This is used to calculate an updated density distribution depending on the stiffness matrix and displacements from the FEA before by determining the Lagrangian parameters which fulfill the optimality criteria. It is a sub-gradient based approach. To include the new density distribution to the FEA in the next iteration step, different heuristic approaches will presented in following paragraphs.

4.1.1 SIMP - Solid Isotropic Material with Penalization

SIMP is the abbreviation form for "Solid Isotropic Material with Penalization". It penalizes densities which are not zero or one by connecting the density and the Young's Modulus proportional with a penalization factor p using a heuristic empirical material law

$$E = \rho^p E_0, \tag{72}$$

where E is the updated Young's Modulus, ρ is the element's density and E_0 equals an initial value of the Young's Modulus. In this report E_0 is always one. The factor p is a user-defined option. In general, the value of p is in a range between three and five. Figure 4.1.1 shows the effect of different penalization factors to densities between zero and 1.

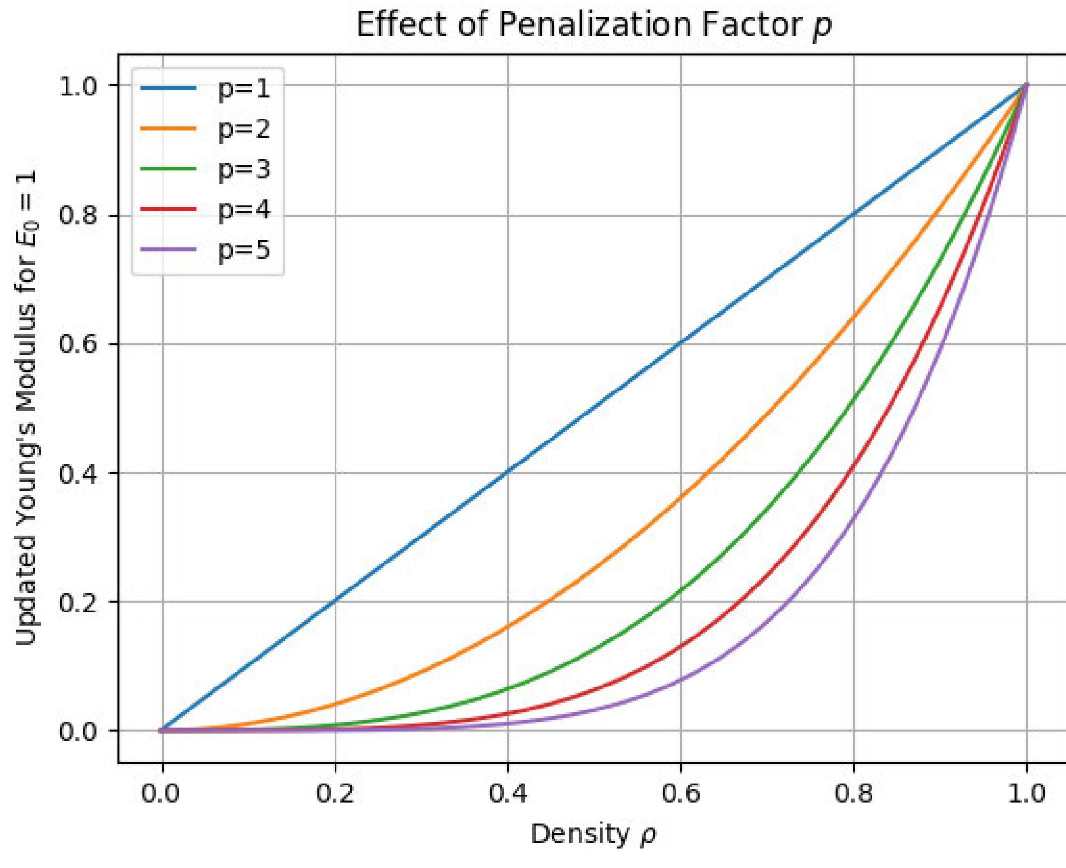


Figure 10: Effect of Penalization Factor for SIMP Approach

To improve the numerical stability, equation 72 is adjusted with a lower bound for the Young's Modulus

$$E = E_{min} + \rho^p (E_0 - E_{min}). \tag{73}$$

This report focuses on the SIMP approach to investigate the feasibility. Further investigations could examine the usage of other approaches to solve topology optimization problems using column generation methods.

4.1.2 RAMP - Rational Approximation with Material Properties

"RAMP" is the abbreviation form for "Rational Approximation with Material Properties". This approach provides similar results as using "SIMP" but the relation between the normalized density and Young's Modulus is different

$$E = E_{min} + \frac{\rho}{1 + q \cdot (1 - \rho)} \cdot (E_0 - E_{min}). \tag{74}$$

The factor q is equivalent to the penalization factor p for the SIMP approach. This equation also includes the lower bound of the Young's Modulus E_{min}. The following figure also shows the impact of the penalization factor q.

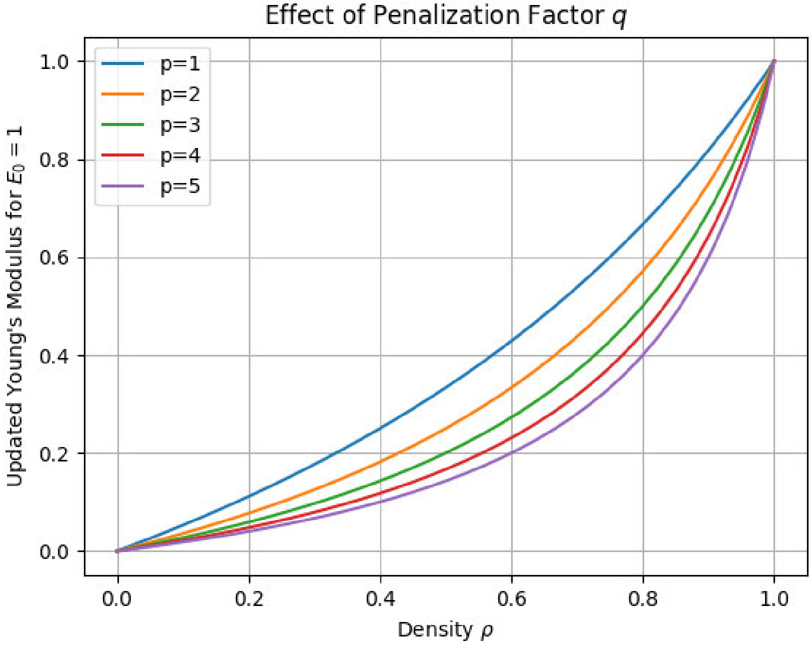


Figure 11: Effect of Penalization Factor for RAMP Approach

4.2 SKO - Soft-Kill-Option

The "Soft-Kill-Option" approach (SKO) differs from the OC approaches as it attempts to imitate geometries and structures that can be found in nature. This basic idea is based from the assumption that the structures are almost ideal due to the long evolution process. There is no explicit mathematical background such as the Lagrangian equation and the corresponding optimal criteria. Another difference between those two optimization processes is that the OC minimizes the compliance (maximizing the stiffness) and the SKO attempts to maximize the strength of a geometry. This is a non-sub-gradient-based approach. An advantage of this genetic algorithms is the fast performance to receive results.

4.3 TOSS - Topology Optimization for Stiffness and Stress

The last presented approach in this report is "Topology Optimization for Stiffness and Stress" (TOSS). It combines the OC and the SKO approach, making it a multi-objective optimization method. Firstly, it uses an OC approach to optimize the stiffness. After that, the structure's strength is maximized using the SKO approach. One advantage of this method is the homogenization of the surface tension which is more close to reality.

4.4 Filter

During the optimization process, an unwanted checkerboard effect may occur (Figure 12).

Possible Checkerboard Pattern Effect

Nodes

Figure 12: Possible unwanted Checkerboard Effect

Using high order elements (e.g. shape functions with quadratic order) or applying a filter can reduce this effect. The report considers just the filter method and the current implementation just contains a common sensitivity filter. The filter is applied before the Lagrangian equation for the optimality criterion is solved. It updates the partial derived compliance of an element regarding its density using weight factors which depend on neighboring elements. The neighboring elements are defined by an impact radius r_{min} and the distance d between two elements. The following equations describe the

implemented filter [SS13]:

$$\widetilde{\frac{\partial c}{\partial \rho_e}} := \frac{1}{\rho_e} \sum_{j \in M_n} \frac{\partial c}{\partial \rho_e} w_{ej} \rho_e \tag{75}$$

$$W := (w_{ij}) \in \mathbb{R}^{n_e \times n_e} \tag{76}$$

$$w_{ij} := \begin{cases} \frac{r_{min} - d(e,j)}{\sum_{\epsilon \in M_n} (r_{min} - d(e,\epsilon))} & : j \in M_n \\ 0 & : j \notin M_n \end{cases} \tag{77}$$

$$M_n := \{e, j \in M | d(e,j) \leq r_{min}\} \tag{78}$$

The equations above show a minor adjustment by defining a weights matrix W but it is the same formulation as in the cited paper. Sometimes the weights matrix W is called "convolutional operator" in literature. The paper contains several other filters for topology optimization problems, which can be interesting to improve the results of this report, so it is suggested to examine if another filter performs better for these kinds of problems. Furthermore it could be interesting to use high-order elements for the CG approach.

4.5 Solving TO Problem with SIMP Approach

This section outlines the original TO problem including the SIMP approach. The previous sections are summarized to clarify which formulation is assumed when the new approach is developed.

$min:$	$c = \vec{f}^T \cdot \vec{u}$	Minimize the compliance
$s.t.:$	$\boldsymbol{S}(\underline{\rho})\vec{u} - \vec{f} = 0$	Elasticity condition must be fulfilled at any time
	$\dfrac{V(\underline{\rho})}{V_f} \leq v_f$	Reach a pre-defined target volume
	$\boldsymbol{S}(\underline{\rho}) = \sum_{e \in M} \rho_e^p \cdot \boldsymbol{S}_e$	Apply SIMP-Approach
	$0 \leq \rho_e \leq 1,\ e \in M$	Consider normalized densities
	$M = \{1, 2, 3, \ldots, n_e\}$	Define set of elements
	$N = \{1, 2, 3 \ldots, n_n\}$	Define set of nodes
	$D = \{1, 2, 3 \ldots, n_d\}$	Define set of degrees of freedom
	$\vec{u} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n_d} \end{pmatrix} \quad \underline{\rho} = \begin{pmatrix} \rho_1 \\ \rho_2 \\ \vdots \\ \rho_{n_e} \end{pmatrix}$	Define vectors for design variables

4.6 Algorithms using SIMP Approach

This section deals with the algorithm which is implemented to solve any kind of topology problem in this report. It uses the OC method with the SIMP approach and needs an initial density distribution. Before the specific algorithm is shown, a table displays the user-defined parameters of the algorithm except the initial density distribution.

Parameter	Symbol	Default	Note
Penalization Factor	p	3	Penalization Factor of SIMP-Approach
Neighboring Radius	r_{min}	1.5	Radius used for Sensitivity Filter
Damping Factor	η	0.5	Damping Factor used for solving Lagrangian Equation

Table 4: Mutable Parameters

The parameter r_{min} strongly depends on the size of the elements and the distance between the centroids of two elements. As long as the elements are square and have an edge length of one, the default value of 1.5 provides that every element is surrounded by neighbors.

Following, the general algorithm to solve a topology optimization problem using the OC approach with the SIMP heuristic is shown.

Algorithm 1 Solve TO-Problem using SIMP-Approach

function SOLVE($\underline{\rho}_{init}$)

  $b \leftarrow 1$

  $\underline{\rho} \leftarrow \underline{\rho}_{init}$

  while $b > 1e^{-3}$ **do**

    $\vec{u} \leftarrow \text{FEA}(\boldsymbol{S}(\underline{\rho}, p))$

    $c \leftarrow \vec{u}^T \cdot \vec{f}$

    $\widetilde{\frac{\partial c}{\partial \underline{\rho}}} \leftarrow \text{SensitivityFilter}(\frac{\partial c}{\partial \underline{\rho}}, r_{min})$

    $\underline{\rho}* \leftarrow \text{updateByOC}(\underline{\rho}, \eta)$

    $b \leftarrow ||\underline{\rho} - \underline{\rho}*||$

    $\underline{\rho} \leftarrow \underline{\rho}*$

  end while

  return $\underline{\rho}*$

end function

The algorithm for the optimizer to update the densities finds the missing Lagrangian multiplier of equation 70 which fulfills the volume constraint [Sig] is outlined below. It is identical to the cited paper. There is also a Python variant of the Matlab code.

Algorithm 2 Optimize Density Distribution rel. to FEA-Results

function UPDATEBYOC($\underline{\rho}$, $\widetilde{\frac{\partial c}{\partial \underline{\rho}}}$, η)
  $l_1 \leftarrow 0$
  $l_2 \leftarrow 1e^5$
  $m \leftarrow 0.2$
  while $l_2 - l_1 > 1e^{-4}$ **do**
    $\tilde{l} \leftarrow \frac{l_2+l_1}{2}$
    $\underline{\rho}* \leftarrow \max(0.001, \max(\underline{\rho} - m, \min(1, \min(x + m, \underline{\rho} \cdot (\frac{-\partial c}{\partial \underline{\rho} \cdot \tilde{l}})^{\eta}))))$
    if $V(\underline{\rho}*) - V_f > 0$ **then**
      $l_1 \leftarrow \tilde{l}$
    else
      $l_2 \leftarrow \tilde{l}$
    end if
  end while
  return $\underline{\rho}*$
end function

These algorithms are used to solve any topology optimization problem in this report because the considered geometries are very simple in this report and it is known that the presented SIMP algorithm can solve these problems reliably. Furthermore, the implementation of this solver is rather simple, given that the solver not complex. In addition, the results can be compared to existing solvers. Further information on the implementation is provided in section 4.7. The presented SIMP solver does not contain any completely new developments and, with the exception of the filter, it is the same as presented in "A 99 line topology optimization code written in Matlab" by O. Sigmund [Sig]. In section 8, some suggestions for other algorithms and methods are highlighted which can be tested whether they perform faster or lead to better results.

4.7 `SIMPProblem` Object

Every `TOModelBase` object and `SubDomain` object contains a `SIMPProblem` object for solving the corresponding domain's topology optimization problem using the SIMP approach. This object is created during instantiating the model object and it maps the topology optimization problem for both a main domain and/or a sub-domain. In addition to the most important function `solve`, which specifically solves a topology optimization problem, there are some other helpful properties and methods. There is also an output class `SIMPOutput` to visualize the optimization (section 9).

4.7.1 Constructor

To create an instance of this class, the node number, the domain's element dictionary and the corresponding finite element model (`FEModel`) need to be passed to the constructor. It also has the task to create the weights matrix for the sensitivity filter (section 4.4) depending on the predefined radius.

4.7.2 Attributes

The class contains all parameters required for the SIMP approach as class attributes. The public attributes can be found in the table below.

Attribute	Type	Short Description
NNodes	int	Node number
Nodes	list	List with node identifiers
Dofs	list	List with DoF identifiers
Elements	dict	Points to element dictionary, which is already known
volfrac	float	Percentage target volume v_f
rmin	float	Radius to determine neighbors r_{min}
loop	int	Main loop counter
Emin	float	Minimal Young's Modulus for numerical stabilization
Emax	float	Maximum Young's Modulus for numerical stabilization; always one
p	int	Penalization factor for SIMP approach
filter	str	Defines which filter is used; just 'sensitivity' available yet
max_main_ iterations	int	Maximum allowed iteration in main loop
H	numpy.array	Weights Matrix / Convolutional Factor
FE_Model	FEModel	Required FE model for corresponding domain
out	SIMPOutput	Output object for optimization visualization
save_out	bool	Save the output figure if true
shape	tuple	Two dimensional design space, just required for an output with rectangular elements

Table 5: Attributes of SIMPProblem class

4.7.3 Function solve

The most important public function is `solve` as it solves the topology optimization problem depending on an initial density distribution, which is the only required parameter for this function. It returns the optimized density distribution, the corresponding displacements, and the resulting value of the objective function, i.e. the compliance. The activation of the output is controlled by the optional parameter `output`, typed as boolean with `False` as default value. If it is `True`, the user will see how the geometry is optimized. Briefly summarized, this function starts by updating the densities of the element objects regarding the initial density distribution and initializes the `change` parameter with one.

```
change = 1
for i, (e_id, element) in enumerate(self.Elements.items()):
        element.x = x_init[i]
```

After that, the main while loop starts by checking whether the predefined main iterations are reached and whether the change is greater than 0.03. Each iteration starts with increasing the counter `loop`. Following, the elements' stiffness matrices are updated by calling the element's object function `update_stiffness_matrix`.

```
# update elements stiffness matrices
for i, (e_id, element) in enumerate(self.Elements.items()):
        element.update_stiffness_matrix(e_min=self.Emin,
            e_max=self.Emax, p=self.p)
```

The matrices are stored in the corresponding element's object, such that the densities. In the next step, the domain's stiffness matrix is calculated and the boundary conditions are incorporated as required. Now, it is possible to perform the FEA to receive the displacements globally.

```
# update domain's stiffness matrix
self.FE_Model.K = self.FE_Model.get_k()
# inc. boundary conditions to new K
if len(self.FE_Model.boundary_conditions.fixed_nodes) > 0:
        self.FE_Model.K, self.FE_Model.F =
            self.FE_Model.incorporate_boundary_conditions(self.FE_Model.K,
            self.FE_Model.F)
# solve and get u
u, self.feasible_solution = self.FE_Model.solve()
```

Each element object has an attribute to store their displacements itself and those attributes are updated by the global displacements. Once the displacements are stored in the objects, the element objects determine its compliance and derive it according to the density.

```
# update elements properties
for e_id, element in self.Elements.items():
        element.update_element_displacements(u,
            np.array(self.FE_Model.global_dofs))
        element.update_flexibility()
```

```
            element.derive_flexibility(p=self.p, e_max=self.Emax,
                e_min=self.Emin)
```

The global displacements are used to calculate the value of the objective function that is just used for the output. After that, the filter is applied to each element. For this, the element object contains a function called `apply_sensitivity_filter` which requires the domain's elements as a dictionary and the corresponding row from the weights matrix. This function updates the element object's property `dCe_filtered` which stores the derived filtered compliance.

```
        # apply filter for each element
        for j, (e_id, element) in enumerate(self.Elements.items()):
                element.apply_sensitivity_filter(self.Elements,
                    self.H[j])
```

In the next step, the density distribution is updated using the OC approach calling a separate function, which is described in the next section. The final steps in one iteration of the main loop update the densities in the element objects and calculate the new value for the parameter `change`

```
        # update elements densities
        for i, (e_id, element) in enumerate(self.Elements.items()):
                element.x = x_new[i]
        # update change
        change = np.linalg.norm(x_new - x_old, np.inf)
```

The complete implementation can be found in the repository as there are also some supporting tools for the output of the process.

4.7.4 Function `__oc__`

The private function `__oc__` optimizes the density distribution depending on the provided derived compliances which indirectly are based the FEA, using the OC approach. The implementation corresponds to algorithm 2, is part of the `SIMPProblem` class and was adopted from Python variant of the 99 line Matlab code [Sig]. As input parameters, the function requires the density distribution, which are to be optimized, and the corresponding filtered derived compliances. It returns the optimized densities. In general, the function tries to find the missing Lagrangian variable, which solves equation 71. During the development of the approach presented in this report, it turned out that this function can cause some issues for several solver inputs, so that this function may need to be revised and made more stable. Zero division can occur under certain conditions, so that there will be an extra error handler for this issue. If the error raises, the objective function will be infinite, the densities and the displacement will be zero. Furthermore, a variable will map the infeasibility of the problem as a boolean. It is not fully clear whether it

is possible to determine solver inputs that raise the zero division error. It is suggested to check it.

4.8 Results for Experimental Model

In this section, the implemented solver optimizes the experimental model, using the standard SIMP approach. Before the results are presented, the applied configuration and initial values are presented briefly.

4.8.1 Configuration and Start Conditions

The configuration parameters to solve the experimental model using the SIMP approach are

- Penalization Factor p: 3
- Target Volume Percentage v_f: 0.44
- Damping Factor η: 0.5
- Filter: Sensitivity
- Neighboring Radius r_{min}: 1.5

, provided the initial density distribution is homogeneous, so each element has the same density in the beginning.

4.8.2 Results

The figure below outlines the results of the topology optimization problem for the experimental model using the standard SIMP approach.

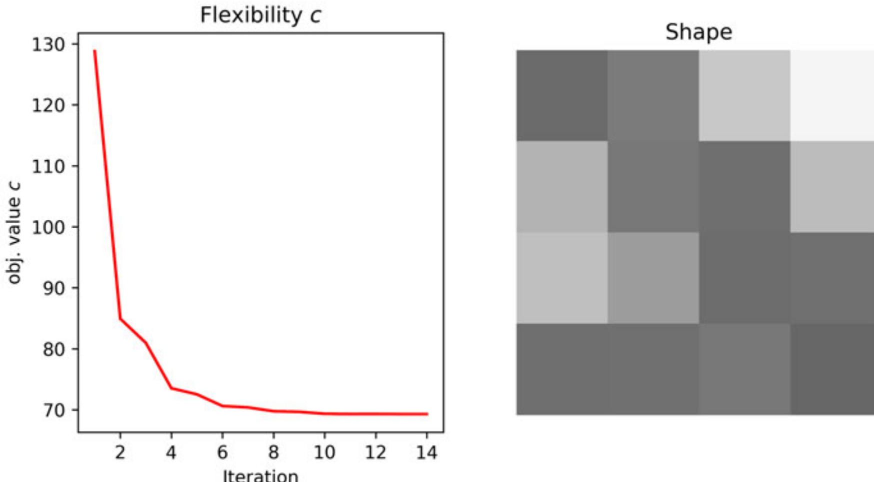


Figure 13: Results of the TO problem of the experimental model using standard SIMP approach

In initially state, the geometry's compliance was 128.797 Nmm. This is improved to 69.297 Nmm during the optimization within 14 iteration steps. This is a reduction of around 46.197%. An application-orientated result would be a real binary solution for the elements, so that an element is clearly switched on or off. The resulting shape, which is shown in the figure above, shows that this purely binary solution is not reached and it is still necessary to analyze the results to find the final resulting shape. Since the SIMP approach considers topology optimization problems continuously (NLP Problem), it is difficult to come to such an application-oriented solution. With a view to performance, the continuous consideration quickly leads to reasonable results, which is one big advantage of this approach, especially if the models become very large. The biggest disadvantage of this approach is the heuristic characteristic so that the problem strongly depends on initial values and is only solved locally. Furthermore, the user needs to guess a reasonable target volume as initial condition.
Briefly summarized, a user would like to reduce the volume of a geometry to a target percentage and find the corresponding optimal shape regarding a given load case. The optimizer adopts the geometry and its initial percentage volume as input. Now, it reduces the compliance by determining the optimal density distribution without increasing the volume.

4.8.3 Validation

As the implementation contains its own variant of the SIMP approach, it is necessary to validate the results to ensure that the implementation works correctly. For this, the SIMP problem for the experimental model is solved again using the Python variant from "A 99 line topology optimization code written in Matlab" [Sig], which is also called "topopt", and the Python module "topy" [Hun+17]. In the table below, the results are compared. All input parameters are the same.

	Decogo TO-Layer	topopt	topy
Flexibility [Nmm]	69.297	70.070	64.914
Iterations	14	10	100
Density Distribution			

The results of all three solvers are similar, so it can be assumed that the implemented solver works correctly. There were several reasons for implementing this rewritten SIMP solver. One reason is the preparation for arbitrary meshes as the other two solvers include some great simplifications regarding usable meshes, elements and geometries. Another reason was the idea of creating a perfect working interface for Decogo so that the solver can be easily adapted to the special requirements of Decogo. In addition, the implementation contributed significantly to the understanding of the mode of operation, although there are several more optional features which can be implemented.

5 Column Generation for TO Problems

This section deals with the column generation (CG) algorithm and its application to TO problems. First, section 5 presents the classic CG method (5.1) before it is connected to the reformulated block-separable TO problem (5.2). After that, the required sub-problems are defined and described (5.3). Lastly, the algorithm for the new approach is outlined (5.4).

5.1 Classic Column Generation

The classic CG method tries to solve a large complex original problem (OP)

$$\begin{aligned} min \ & \sum_{i \in I} c_i^T x_i \\ s.t. \ & \sum_{i \in I} a_i^T x_i \geq b_i \\ & \forall i \in I \end{aligned}$$

using a simple linear master problem (MP)

$$\begin{aligned} min \ & \sum_{i \in \tilde{I}} c_i^T x_i \\ s.t. \ & \sum_{i \in \tilde{I}} a_i^T x_i \geq b_i \\ & \forall i \in \tilde{l}, \ \ \tilde{I} \subset I, \end{aligned}$$

where the size of $\tilde{I}$ is much smaller than the size of I. The master problem, sometimes called inner approximation, is simple to solve globally, e.g. using the Simplex approach, to obtain an optimal primal point $\tilde{x}^* \in \tilde{I}$ and the corresponding optimal dual values μ. To test whether a solution from the MP is optimally related to the OP as well, the pricing sub problem

$$\begin{aligned} \delta &= min\{\delta_i | i \in I\} \\ \delta_i &:= c_i - \mu^T a_i, \quad i \in I \end{aligned}$$

can be analyzed. If $\delta \geq 0$, the point $\tilde{x}^*$ solves the master problem optimal and the objective of the master problem can not be improved anymore. The reduced costs δ can be considered as a converge criterion, or rather as a

potential for improvement of the master problem. If $\delta < 0$, a better solution exists regarding the OP and the MP is updated in accordance to the dual values

$$\tilde{I} = \tilde{I} \cup argmin_{i \in I}\{c_i - \mu^T a_i\}.$$

The updated master problem has to be solved again until all reduced costs are greater than zero. Below, the basic algorithm for CG is presented.

Algorithm 3 Basic Column Generation Algorithm

```
δ ← −∞,  Ĩ ← initColumns()                          ▷ Initialization
while δ < 0 do                 ▷ Check if minimal reduced costs lesser than 0
    x̃, μ ← solveMP(Ĩ)         ▷ Obtain x̃ and corresponding dual values μ
    δ = min{c_i − μ^T a_i | i ∈ I}          ▷ Determine minimal reduced costs
    if δ < 0 then                          ▷ Check if MP has to be updated
        Ĩ ← Ĩ ∩ argmin_{i∈I}{c_i − μ^T a_i}                  ▷ Update the MP
    end if
end while
y* ← solveOP(x̃)                          ▷ Solve OP, using point from MP
```

The master problem contains slack variables which can be analyzed to determine the violation of the global constraints. If all of these slack variables equal zero, the master problem does not violate the global constraints P. Slack variables resemble the artificial added variables in the Simplex algorithm. Further information on CG can be found in the presentation "Column Generation, Dantzig-Wolfe, Branch-Price-and-Cut" by M. Lübbecke [Mlc].

5.2 Apply CG on Block TO

In this section, the CG method is combined with the block TO problem (2.2). In general, the block-separated TO problem with $|K|$ blocks is given by

$$\begin{aligned} & min \;\; c^T x \\ & s.t. \;\; x_k \in X_k, \;\; \forall k \in K, \\ & \qquad x \in P \end{aligned} \tag{79}$$

where P denotes the global constraints, the copy and volume constraints. X_k maps the local block-specific constraints. CG solves a problem like

$$\begin{aligned} min \;\; & c^T x \\ s.t. \;\; & x_k \in conv(X_k) \\ & x \in P \end{aligned} \tag{80}$$

with the corresponding generated master problem (inner approximation)

$$\begin{aligned} min \;\; & c^T x \\ s.t. \;\; & x_k \in conv(R_k), \\ & x \in P \end{aligned} \tag{81}$$

where R_k are inner points of X_k. Using the block-separable formulation of an optimization problem changes the CG algorithm as follows.

Algorithm 4 CG for block-separated notation

- $\delta \leftarrow -\infty,\;\; R \leftarrow initCols()$ ▷ Initialize reduce costs and a Column set
- **while** $\delta \leq 0$ **do** ▷ Check if reduce costs less than 0
  - $x, d \leftarrow SolveMP(R)$ ▷ Solve simple MP depending on R
  - **for** $k \in K$ **do** ▷ Gen. new cols., using sub-problem solutions
    - $y_k \leftarrow argmin\;\; d_k^T A_k x_k,\;\; x_k \in X_k$ ▷ Solve sub-problem
    - $\delta_k \leftarrow min\{d_k^T r_k - d_k^T A_k y_k : r_k \in R_k\}$ ▷ Min. reduced costs for block
    - **if** $A_k y_k \notin R_k$ **then** ▷ If col. is not in current block column set
      - $R_k \leftarrow R_k \cup \{A_k y_k\}$ ▷ add column
    - **end if**
  - **end for**
  - $\delta \leftarrow min\{\delta_k : k \in K\}$ ▷ Get minimal reduced costs of all blocks
- **end while**
- $(Y^*, X^*) \leftarrow SolveOP(x)$ ▷ Solve OP using point from MP

The resource matrix A_k is used to transform points of the sub-problems to the space of the master problem, whereby the matrix contains the coefficient of the objective function c_k and the coefficients of the global constraints. d

denotes a search direction for new columns, including the dual values μ of the master problem. By using the block-separated formulation, it is possible to use solutions of sub-problems to determine the lowest reduced costs. Further information on the master problem, resources and on solving a general optimization problem using column generation methods can be found in the paper "A column generation algorithm for solving energy system planning problems" [Mut+21].

5.3 Sub-Problems

In general, sub-problems for a block k for the CG algorithm above are defined by

$$min\ d_k^T x_k$$

$$s.t.\ x_k \in X_k,$$

where d_k is the search direction provided by the solution of the master problem related to the sub-problem space and is typically defined by its dual values. Regarding the block TO problem, a corresponding sub-problem can be formulated as a TO problem itself.

$$\begin{aligned} min\ & d_k^T u_k \\ s.t.\ & S_k(\rho_k) u_k = d_k \\ & \rho_{ki} \in \{0,1\},\ \ i \in M_k \\ & u_{kd} \in \left[u_{kd}^-, u_{kd}^+\right],\ \ d \in \tilde{D}_k. \end{aligned}$$

Furthermore, the sub-problems are created by applying the domain decomposition (3.8) to the original model. The block-specific partial direction d_k should correspond to the outer forces of the corresponding sub-domain f_k. As the master problem does not consider the boundary forces g (3.8.5), they have to be incorporated to the sub-problems to avoid ill-posed problems. The sub-problems in this approach are solved by a provided initial value from the MP, so the volume constraint for a block can be omitted because the initial point defines the target volume for the OC approach and depends on the global volume constraint. Using the boundary forces leads to the following

definition of sub-problems for a block TO problem.

$$
\begin{aligned}
min\ & (d_k + g_k)^T u_k \\
s.t.\ & S_k(\rho_k) u_k = d_k + g_k \\
& \rho_{ki} \in \{0, 1\},\ \ i \in M_k \\
& u_{kd} \in \left[u_{kd}^-, u_{kd}^+\right],\ \ d \in \tilde{D}_k.
\end{aligned}
$$

In this report, these sub-problems are solved by the SIMP approach to receive first results and determine the feasibility and usefulness of this approach. To improve the quality of the results further, it is suggested to develop a global solver for small TO problems e.g. using the "SCIP - Optimization Suite"[Bes+21].

5.3.1 Algorithm

Below, the algorithm for solving a TO sub-problem by the SIMP approach is shown in more detail. It depends on an initial value of the master problem $\underline{x}_{k,init}$ and the partial direction d_k related to the corresponding block k.

Algorithm 5 Sub-problem solving related to CG

function SolveTOSubProbl($\underline{x}_{k,init}, d_k$)

  $\underline{\rho}_k, \vec{u}_k, \tilde{f}_k \leftarrow Split(\underline{x}_{k,init}, d_k)$

  $\vec{g}_k \leftarrow GetBoundaryForces(\underline{x}_{k,init})$

  $\tilde{f}_k \leftarrow \tilde{f}_k + \vec{g}_k$

  $\underline{\rho}_k^*, \vec{u}_k^*, c_k^* \leftarrow SIMP(\underline{\rho}_{k,init}, \tilde{f}_k)$

  $s \leftarrow Check(\vec{u}_k^*)$

  return $\underline{x}_k^*, c_k^*, s$

end function

The function `Split` decomposes the initial values and the partial direction related to the density and displacement variables. The vector $\tilde{f}_k$ replaces the sub-domain's force vector $\vec{f}_k$ so that the sub-problem is solved depending on the direction of the master problem. s denotes a boolean value, which maps the feasibility of the solution provided by the SIMP approach. As long as the SIMP approach is completed without any errors, this value is `True`.

5.4 TOCG algorithm

This section shows the algorithm for solving TO problems using column generation methods applied to the block-separable formulation.

Algorithm 6 Block-TO CG

$\delta \leftarrow -\infty,\ \ R \leftarrow initCols()$ ▷ Initialize reduce costs and a Column set
while $\delta \leq 0$ **do** ▷ Check if reduce costs lesser than 0
  $x, d \leftarrow SolveMP(R, P)$ ▷ Solve simple MP depending on R and P
  for $k \in K$ **do** ▷ Gen. new cols., using sub-problem solutions
    $g_k \leftarrow$ GetBoundaryForces(x) ▷ Get g_k related to sol. of MP
    $X_k \leftarrow$ updateLocCons(g_k) ▷ Update the local constraints X_k
    $y_k \leftarrow argmin\ \ (d_k + g_k)^T u_k,\ \ x_k \in X_k$ ▷ Solve sub-problem
    $\delta_k \leftarrow min\{d_k^T r_k - d_k^T A_k y_k : r_k \in R_k\}$ ▷ Min. reduced costs for block
    if $A_k y_k \notin R_k$ **then** ▷ If col. is not in current block column set
      $R_k \leftarrow R_k \cup \{A_k y_k\}$ ▷ Add column
    end if
  end for
  $\delta \leftarrow min\{\delta_k : k \in K\}$ ▷ Get minimal reduced costs of all blocks
end while
$(Y^*, X^*) \leftarrow SolveOP(x)$ ▷ Solve OP using point from MP

The algorithm shows that it is different from the classic column generation algorithm as the local constraints X_k are not static and changed in every iteration. This is comparable with a trust region approach, with the difference the trust region is continuously updated. It is necessary to keep in mind that this is just a first approach to determine the usefulness and feasibility of this new approach. There are several ongoing research projects and suggestions to improve the performance and quality, which would exceed the scale of this report. Below, the implementation is outlined and some first results are presented.

6 Implementation and Results

This section mainly presents the second part of this report: the development of the TO layer for Decogo. Furthermore, first results for solving a TO problem using the new column generation approach are shown. In addition, this section also highlights some small adjustments to improve the quality of the solutions. The section starts by giving an overview of the basic course of action of the implementation and providing the general implementation structure. After that, the base class for a TO domain is shown, before two procedures of creating the original TO problem are shown. Then, Decogo and its generic framework is presented briefly. Afterwards, the explicit TO layer for Decogo is shown. Using this implementation, first results are generated and shown then. Lastly, this section introduce some small extensions of the TOCG algorithm.

6.1 Basic Course of Action

This subsection outlines the basic course of action for solving a TO problem with the new approach using the developed implementation. This is briefly done by the following list.

1. Create an original structure/geometry/model and formulate TO problem
2. Reformulate in accordance to block-separable formulation
3. Generate sub-problems
4. Pass it to Decogo

6.1.1 Implementation Structure

This section gives a first overview of the developed implementation, including a visualized course of action of the implementation. The total folder structure can be found in the Annex (A.1). The actual functionality for the solver is located in the folder "pylib" located in the root dictionary. The current implementation includes several features for the visualization of the results, which will briefly be presented in section 9.
The following figure outlines the general course of action of the implementation.

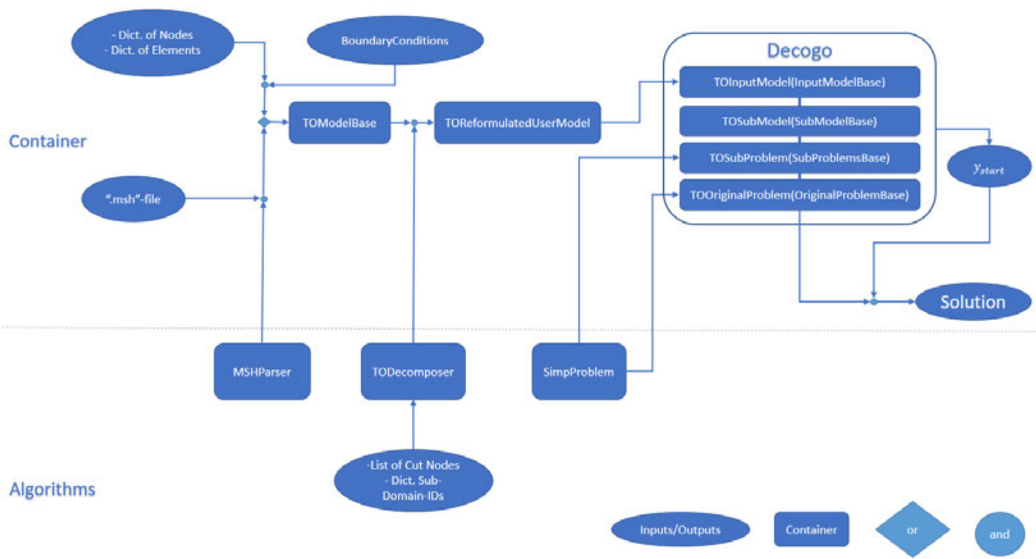


Figure 14: General course of action of the implementation to solve a TO problem with new approach

There are two different ways to create an original model object. This will be reformulated before it is passed to Decogo. The names of the containers in the figure correspond to the class names of the implementation.

6.2 `TOModelBase` Object

The `TOModelBase` class represents an original domain for a topology optimization problem. It provides general information on the domain as well as the corresponding finite element model and the optimization problem. The class is placed in "pylib/tomodel/tousermodel.py".

6.2.1 Instantiate

To create an instance of the `TOModelBase` class, the constructor's parameters are shown in the table below.

Parameter	Type	Description	Required?
`nodes`	`dict`	Keys describe the global node-id; Value is the corresponding `Node` object	Yes
`elements`	`dict`	Keys describe the global element-id; Value is the corresponding `Element` object	Yes
`boundary_ conditions`	`Boundary Condition`	Predefined `BoundaryCondition` object for the original domain	Yes
`dofs_per _node`	`int`	Number of degrees of freedom at a node	Yes

With these parameters it is possible to define a domain for a topology optimization problem. There are two procedures to create this object which are presented in the next subsection. The constructor also solves the domain's FEA with the initial values to generate first values for the displacements.

6.2.2 Attributes

In general, the attributes of the `TOModelBase` correspond to the symbols of table 1, but there are some minor differences in the implementation, e.g. implemented objects for smarter usage. The property `Nodes` is the same as the input parameter `nodes` and the property `elements` is the same as the input parameter `elements`. The input parameter `boundary_ conditions` is required to define an object which maps the boundary conditions for the corresponding domain (3.4.3). Furthermore, the constructor of `TOModelBase` creates another object which maps the topology optimization problem. It will also be used to solve the problem for the corresponding domain.

6.2.3 Methods

The class itself does not contain a large number of methods. One function is `reformulate`, which creates a reformulated model by using a specified "decomposer" and is always required for the layer for Decogo. To calculate the volume of a domain, the function `get_volume` can be used which is shown below.

```
    def get_volume(self):
        return sum(e.x * e.V for i, e in self.elements.items())
```

In addition, the class provides the function `display(self)` to print all information of a domain to the console. It is just a supporting function and is not absolutely necessary.

6.2.4 `TOReformulatedModel` object

An object of the class `TOReformulatedModel` maps the reformulation of the TO problem (2.2.9). It provides information on the sub-domains, or rather the blocks, and prepares information about the required constraints and the design variables. Calling the function `reformulate` of the class `TOModelBase` by passing a `TODecomposer` object to it returns an instance of this class.

6.2.5 `TODecomposer` class

An object of the class `TODecomposer` can be used to perform the presented domain decomposition. It will decompose and reformulate an `TOModelBase` object so that it is prepared for Decogo. For instantiating, a list of the cut nodes numbers, a dictionary of sub-domains and the base model needs to be passed to the constructor. To improve usage, it should be possible to develop an algorithm to decompose a main domain automatically just by providing a list of cut node numbers. The dictionary of sub-domains contains the number of the sub-domain as the key and a "raw" object of `SubDomain` as the corresponding value. The constructor of the `TODecomposer` class updates the sub-domains by inserting the corresponding boundary conditions and setting the initial displacements from the main domain. This class provides some mappings for nodes, DoFs and sub-domains. To apply the decomposer, the class `TOModelBase` provides the function `reformulate` which can be called by passing an instance of the decomposer class to it. The current implementation is not ideal, but it is sufficient for testing the new approach.

6.3 Model Creation

This section outlines the implemented features to create models which can be optimized, using Decogo. In general, there are two basic procedures for the creation: a manual programmatic procedure and a procedure using the finite element mesh generator "Gmsh" (https://gmsh.info/) [GR09]. Both procedures return a `TOModelBase` object, which is used for further steps.

6.3.1 Programmatic

The constructor requires two dictionaries for the nodes and for the elements and a `BoundaryCondition` object for the domain, which can be created by manually. The Annex (A.2) provides an example for a manual model. Briefly summarized, lists of identifiers for the nodes and the elements are created using the `range` object. The coordinates for the Nodes have to be created manually. During the creation of the element objects, it is necessary attention must be paid to the order of node numbers. As long as the model is simple, like in the example, this procedure can be an alternative to the other procedure, as this one can be very fast and more stable and does not show unwanted side effects. If the element formulations are more complex or the size of the problem is larger, the procedure using "Gmsh" is suggested.

6.3.2 Using "Gmsh"

Application-oriented meshes are complex and large in terms of the geometry. To handle complex geometries and mesh them, the open-source tool "Gmsh" can be used to create meshes and export them as a mesh file (.msh). The tool can be used to mesh existing geometries and/or create new ones.

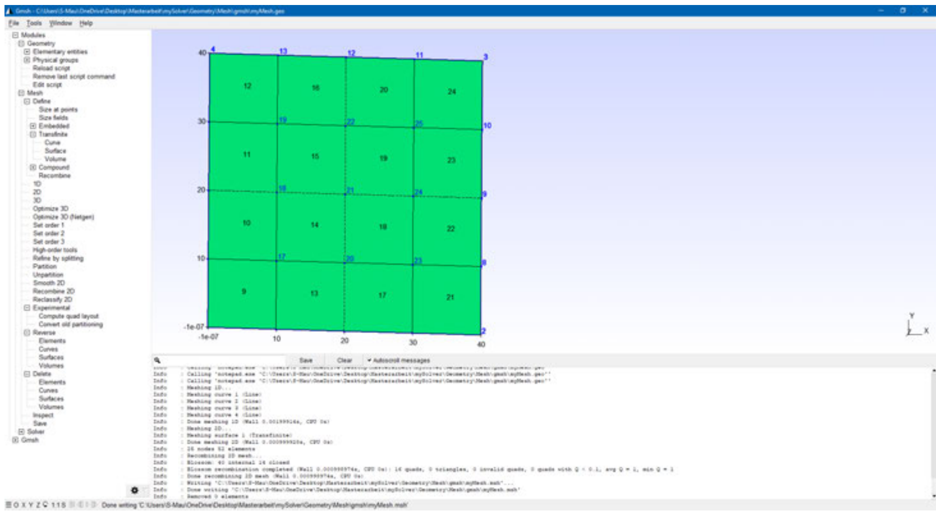


Figure 15: Geometry and mesh created using "Gmsh"

Besides meshing complex geometries, the procedure offers the advantage that most element geometries (triangles, rectangles, tetrahedrons, hexahedrons, ...) can be used. The mesh file also provides the correct node order for each element.

Gmsh - Boundary Conditions It is also possible to determine boundary conditions using the software tool. For this, physical groups need to be added to the geometry. For fixations, the name of the corresponding group must contain "fix", whereas physical groups which describe the load cases must contain "load" in their name.
After creating the mesh, it needs to be exported as a ".msh"-file. The applied options for the export can be found in figure 16. The file always has the same structure. It starts with a header which is followed by different sections for the nodes, the physical groups and elements. An example of a ".msh"-file is provided in the Annex(A.3) and further information on the structure of such a file can be found in the reference manual of "Gmsh" on their website.

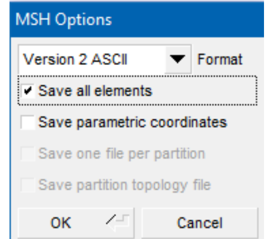


Figure 16: Export options

6.3.3 `MSHParser` class

To use the created mesh file, the implementation provides a parser class, called `MSHParser`. It can read the mesh file and can create a `TOModelBase` object based on the mesh file. The class can be found in "pylib/meshparser/mshparser.py".

Constructor The constructor of the `MSHParser` class just requires the path for the corresponding mesh file. It starts by reading the file and determining the position of the different sections in the file. After that, it translates the file object descriptions to the required object type and stores them as class properties. All of these properties can be found in the implementation. There are also two supporting classes for a ".msh" node and element, but they are just containers to store certain file properties.

Methods The most important public method is `get_base_model`. It returns the `TOModelBase` object which corresponds to the mesh file presented to the constructor. It also takes the optional parameter `_bc` which is used to consider manual boundary conditions, so the user can decide if the boundary conditions are taken from the mesh file or from the code itself. The method `write_mesh_file` writes a ".mesh" file to the same path as the original file.

6.4 Decogo

The solver which is used to apply the CG methods to the TO problem is called Decogo. It was developed in pure Python language at HAW Hamburg. The article "A column generation algorithm for solving energy system planning problems" [Mut+21] provides detailed information on the solver. This section briefly summarizes Decogo's tasks, its special features and modification from the classic CG. Decogo's main task is generating and solving the master problem (inner approximation) for an input model. It generates the master problem using the Pyomo modeling language and solves it by using "Gurobi"[Gur]. The algorithm of Decogo varies from classic CG and is divided into three phases. During the first phase, it initializes columns automatically by a sub-gradient method, where sub-problems are also solved. Phase two can generate a large number of columns by solving sub-problems approximately, or rather locally. Through this, the slack variables should be eliminated. In addition, a heuristic will be provided for this elimination, which is described in section 6.5.5. In phase three, the classic CG is applied to the input model to improve the quality of the results from phase two and solve the sub-problems globally. As this report just considers the feasibility and the usefulness of the new TOCG approach, phase two and three will be similar, as just a SIMP solver is implemented and can be used for solving sub-problems.

6.4.1 Generic Framework of Decogo

Decogo is generic, as it provides a framework for creating user-specific layers. The Frameworks is based on abstract classes and the Python "abc" module. Every layer requires inherited classes from those base classes and the corresponding methods need to be overridden so that the Decogo understands the new layer. Currently, there is a Decogo layer for the Pyomo modelling language [Byn+21] so that Decogo can solve a Pyomo model using column generation methods. The following table summarizes the abstract classes briefly.

Abstract Class Name	**Short Description**
`InputModelBase`	• Passed to Decogo • Contains all information • Instances of all other abstract classes placed here • `cuts` property maps constraints
`SubModelBase`	• Contains variables of a block k • Is more like a container only
`SubProblemBase`	• Provide solvers for sub-problems • Is used while CG
`OriginalProblemBase`	• Provide a solver for the original problem • Provide heuristic for slack elimination

Table 6: Abstract classes of framework

In addition, the next figure shows the UML diagram of the framework. The full information can be found in the specific implementation and the documentation of Decogo.

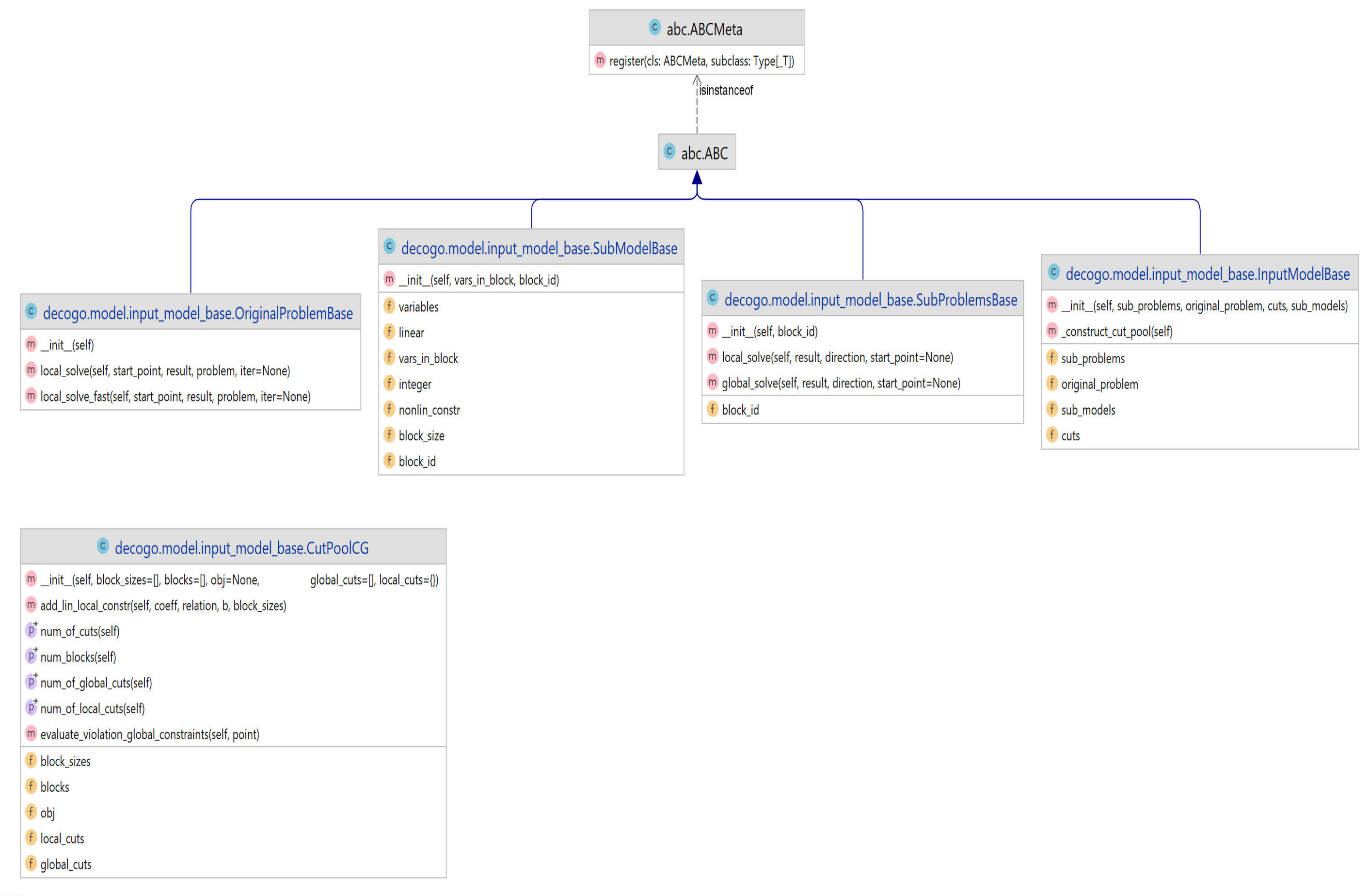


Figure 17: UML diagram of framework

6.5 Decogo TO Layer

In this section, the basic implemented TO layer for Decogo is outlined, including an input model, sub-models for the corresponding sub-domains with the relating sub-problems and the required class to map the original problem. Furthermore, there is an object to hold the constraints of the TO problem.

6.5.1 Input Model

An instance of the class `TOInputModel` can be passed to Decogo. Regarding the presented framework for Decogo (6.4.1), this class inherits from `InputModelBase`. For creating an instance of this class, the constructor takes an object of `TOReformulatedModel` as the only required parameter. It provides all required information so that Decogo can handle this input model. In the following table, the most important properties are listed with a short description and an indicator whether they are abstract and overridden from the base class.

Name	Type	Is abstract?	Description
`model`	`TOReformulated Model`	No	Maps the reformulated model
`settings`	`Settings`	No	Basic Decogo settings
`blocks`	`list[list[str]]`	No	Variables for each block as string
`block_sizes`	`list[int]`	No	Number of variables in each block
`cuts`	`CutPoolCG`	Yes	Maps objective and at least the global linear constraints
`sub_models`	`list[TOSubModel]`	Yes	List with objects for each block
`sub_problems`	`list[ TOSubProblem]`	Yes	List with objects, which map a sub-problem for a block
`original_problem`	`TOOriginalProblem`	Yes	Solves the original problem

The implemented class can be found in the file "pylib/tomodel/to_input_model.py". The TO layer is configured separately from Decogo with a configuration dictionary. The input model also holds some of the possible user defined options, which can be set by calling the object's method `configure` and passing the corresponding dictionary to it. An example dictionary can be found in Annex (A.4) and in the "config.py" file at the top level of the folder structure in the repository.

6.5.2 CG Cut Pool

The `CutPoolCG` class is provided by Decogo and every model requires one object of this class in the input model. The objective function and at least the linear global constraints P are handled and evaluated by objects in this class. The input model contains the private abstract function `_construct_cut_pool` which returns the required parameters to create an instance of `CutPoolCG`.

As the global linear constraints connect the different blocks to each other after an original problem is decomposed, these constraints are called "cuts" and all of these cuts define a pool of constraints, relating to the reformulated problem. For the TO problem, the cuts are the volume constraint and the copy constraints.

6.5.3 Sub-Model

An instance of `TOSubModel` provides the variable definition of a block. In the case of the TO problem, the variables are the densities $\underline{\rho}_k$ and the displacements $\vec{u}_k$ of a block or a sub-domain k. To describe a variable for Decogo, the solver provides a class called `VarDomain`. Further information on it can be found in the documentation of Decogo. The displacements variables are always of type `real` ($\in [0, 1]$) with the bounds described in chapter 3.5.3. The type of density variables depends on the consideration of the TO problem. If the problem is solved as a NLP problem, the variables will be also of type `real`. However, the variables' type will be `int` ($\in \{0, 1\}$), if the TO problem is solved as a MINLP problem. All variables are stored in one list and the corresponding class attribute is called `variables`. This list has the same shape as the block specific summarized variable vector

$$\underline{x}_k = (\underline{\rho}_k, \vec{u}_k)^T$$

The class also contains the property `block_id`, which maps the identifier for the sub-domain and the property `linear`, which can be set to true if a block has only linear constraints. In the case of a TO problem, a block is always non-linear, so this property is always false.

6.5.4 Sub-Problem

The class `TOSubProblem` is used to handle and solve the sub-problems for Decogo. The partial solutions of the sub-problems regarding the original problem are used to generate columns, which feed the inner approximation so that it becomes similar to the original problem. This abstract class contains two abstract methods, which need to be overridden for the TO specific layer. These two abstract methods are `local_solve` and `global_solve`. In this report, only a SIMP solver is used to solve any TO problem as it is sufficient to examine the feasibility of the new approach. In section 8, some processes and other approaches for further research on solving TO sub-problems are presented. The input parameters for these methods are

- a result object provided by Decogo

- the direction d from the master problem
- the point of the master problem according to the direction as the start point for the SIMP algorithm
- a problem object provided by Decogo which stores for example the columns

Like the input parameters, the return is also the same as

- optimized point of the sub-problem
- primal bound
- dual bound
- boolean if the solution is feasible regarding the local constraints

Regarding the primal and dual bound of the TO problem, as long as SIMP is used, it is not possible to calculate the correct dual bound in terms of the global optimality because of the heuristics used in this approach. So the implemented methods set the primal and dual bound equal. If the SIMP approach finds a solution candidate, it will always fulfill the local constraints, as it is an OC approach. In this new approach, it is possible that the SIMP approach runs into an error while evaluating the Lagrangian parameter in the OC optimizer depending on the provided start point and direction, so an error handler is used in the abstract methods, which sets the boolean value for the feasibility to `False` if this error raises. In addition to the error, the objective value and so the primal and dual bounds are set to infinite. These methods are called in each of the three phases of Decogo. Regarding the expected results of the sub problems, Decogo should not find new columns in its third phase, because there is just a local solver available at this point (SIMP solver) so that the local and global solver are the same. The complete implementation of the class `TOSubProblem` is also located in the "pylib/tomodel" folder.

6.5.5 Original-Problem

The last required class for the framework is `TOOriginalProblem`, which inherits from `OriginalProblemBase`. Its two main tasks are solving the original problem related to the start point provided by Decogo and eliminating the slacks from the master problem by finding any feasible point of the original problem. These two tasks are implemented by two abstract methods in the base class which need to be overridden. The function `local_solve`

solves the original problem regarding a calculated point by Decogo. Firstly, the provided point is split into two partial vectors for the density variables and the displacement variables. After that, the original problem is solved by the SIMP approach and a new primal bound is determined. The primal bound is used to test the quality of the master problem. This is done by calculating the gap between the objective value of the MP and the primal bound of the original problem by using the SIMP approach

$$gap := val(80) - val(81). \tag{82}$$

The second function is `local_solve_fast`, which attempts to find a first feasible solution candidate to eliminate the slacks of the master problem. This task can be extremely difficult for complex problems or if the problem is considered as a true MINLP problem. However, for the NLP variant of a TO problem, finding a feasible solution candidate is simple, as the initial values calculated by the initial FEA are feasible. If the TO problem is considered as a MINLP problem, finding a solution could be difficult, as the initial state is a fractional solution with densities between zero and one. Random binary density distribution can easily violate the volume constraint or lead to illogical structures compared to the original problem. Currently, the SIMP approach performs two iterations, the resulting densities are round and the corresponding displacement and compliance is solved again. Some tests show that this process violates the volume constraint, so the right-hand side of this constraint is updated with the rounded values. This procedure is very heuristic and unstable so there is ongoing research on this issue to make that process stable for a MINLP problem without updating the constraint in phase two of Decogo. Another way to remove the slacks from the master problem is generating that many columns so that a random combination of them map the original problem similarly. Maybe, this way is more useful for solving the TO problem as a true mixed-integer problem.

6.5.6 Algorithm

This section presents the general algorithm for solving a block-separable TO problem using Decogo depending on the different phases of Decogo.

Algorithm 7 SolveBlockSeparatedTO

Require: $\underline{x}_{init}$, $phase$ ▷ Initial homogeneous mass distribution

  $n \leftarrow 0$ ▷ Set initial step

  $\vec{u}_{init} \leftarrow FEA(\boldsymbol{S}(\underline{x}_{init}), \vec{f})$ ▷ Get initial $\vec{u}$

  $\underline{y}_{init} \leftarrow (\underline{x}_{init}, \vec{u}_{init})^T$ ▷ Build initial solution vector

  repeat ▷ At least one Step performs

    $d^n, \tilde{y}^n \leftarrow Decogo : getSearchDirections(phase)$ ▷ Call Decogo

    if $n = 0$ **then** ▷ Empty approx. sol. vector for $n = 0$

      $\tilde{y}^n \leftarrow \underline{y}_{init}$

    end if

    $\vec{g} \leftarrow getBoundaryForces(\tilde{y}^n)$ ▷ Update boundary forces

    for $k \in K$ **do** ▷ Consider each block

      $\underline{y}_k^{n+1} \leftarrow SolveSubProblSIMP(\tilde{y}_k^n, \vec{d}_k^n, \vec{g}_k^n)$ ▷ Solve sub-problem

      if $\underline{y}_k^{n+1}$ Is feasible and new point in Y_k **then**

        $Decogo : (S_k, R_k) \leftarrow updateDecogo(\underline{y}_k^{n+1})$ ▷ Update S_k & R_k

      end if

    end for

    $n \leftarrow n + 1$ ▷ Increase step

  until max. step reached ▷ Stop condition

  $\underline{Y}^n \leftarrow Decogo : getLastPoint()$ ▷ Get the last Point from Decogo

  $\underline{Y}^* \leftarrow SolveOriginalTO(\underline{Y}^n)$ ▷ Solve original TO problem with x_n as initial

---

6.6 Solving TO Problems using Decogo

Section 6.6 presents first results of solving the experimental model with the TOCG algorithm using Decogo. It starts by solving the model considering the TO problem as a NLP problem (6.6.1) which should be the most simple variant of the TO problems. After that, the TO problem of the model is attempted to be solved as a MINLP problem (6.6.2).To differentiate between those two variants of solving a TO problem, the configuration dictionary for the TO layer provides the key `MINLP`. The value of the key is of the type `bool`, so that the problem is solved as a MINLP problem if the value is `True`; otherwise the problem is solved as a NLP problem. The initial conditions for applied SIMP solver are the same in both solving variants

- Neighboring radius r_{min}: 1.44
- Fractional target volume v_f: 0.44
- OC damping factor η: 0.5
- Penalization factor p: 3
- Filter: Sensitivity

The configuration of Decogo is also the same for both variants and the main options are

- max. time: 1000 seconds
- CG max. iterations: 5
- sub-gradient max. iterations: 5
- Use fast Frank Wolfe: False
- LP solver: Gurobi [Gur]

The code below shows how an input model can be passed to Decogo, be configured and solved.

```
from decogo.solver.decogo import DecogoSolver  # import solver
from config import config_decogo, Configuration_TOLayer  # import
    configs
from examples.topopt_model import inp_model  # import example input
    model
if __name__ == "__main__":
        solver = DecogoSolver()  # create an instance of the solver
        config_decogo()  # configure Decogo
        inp_model.configure(Configuration_TOLayer) # configure the
            input model
        solver.optimize(inp_model)  # solve the TO problem
```

All log files related to the presented results are provided in Annex (A.5).

6.6.1 NLP

In this section, the experimental model is solved with Decogo as a NLP problem and compared to the original results. Solving the TO problem as a continuous NLP problem deviates from the general problem, but it should provide a first impression for solving TO problems using CG methods. Below, the configuration of the implementation is shown including the SIMP options

```
Configuration = {
        'report_name': NLP,
        'MINLP': False,
        'SimpProblem': {
                'volfrac': 0.44,
                'penalization_factor': 3,
                'emin': 1e-9,
                'emax': 1,
                'rmin': 1.44,
                'filter': 'sensitivity'
        }
}
```

Furthermore, it should be noted that no active or passive element is considered. The problem should be the most easiest possible variant. The next table (table 7) presents some statistics of the RTO problem and the first results of the TO problem solved by Decogo.

Model Statistics		
Number of blocks/ sub-domains		4
Total number of variables		88
	Density variables	16
	Displacement variables	72
Variable numbers in blocks		
	Sub-domain 1	22
	Sub-domain 2	22
	Sub-domain 3	22
	Sub-domain 4	22
Number of global constraint		23
Solver statistics		
Total time [seconds]		25.212
Main iterations		1
Number of CG iterations		2
Number of solved sub-problems		60
Number of generated columns		10
Sum of slacks in MP		0.0
Maximum slack value in MP		0.0
Results		
Primal bound of RTO		68.96
CG relaxation objective value		128.80
Quality gap		46.46%

Table 7: First results for experimental model solved by Decogo

The table above shows that Decogo understands the TO problem and can perform on it. Decogo is able to find columns to feed the master problem

with it. In addition, the slack variables were eliminated by presenting a feasible problem to the master problem. Regarding the third phase of Decogo, the expected results can be explained by the fact that Decogo was not able to find further columns to improve the objective value of the master problem, so the solver converges after one iteration. Nevertheless, the results reveal several challenges which must be mastered to receive useful results by this approach. The algorithm was not able to find better columns which improve the objective value of the master problem. The value is constantly 128.80 after eliminating the slack variables. Due to the results of the original problem, it is known that better results exist with a compliance of 69.257 Nmm. This is also reflected in the quality gap of 46.46%. As long as better columns are not found, the bound can not be decreased or changed at all. It is striking that only ten columns were added to the master problem. This could be a reason for the lack of improvement in the master problem's objective value. Analyzing the starting points presented to the sub-problems reveals that they are often equal or at least similar to points which are used to solve sub-problems before, so that it seems that the current implementation is not flexible enough when it comes to the different provided densities and displacements. This static behavior can also be shown while plotting the initial values from the master problem to solve the original problem to determine a new primal bound (18). It completely corresponds to the initial homogeneous density distribution for calculating the initial values by the FEA at the beginning. This underlines, that the algorithm does not find any better columns.

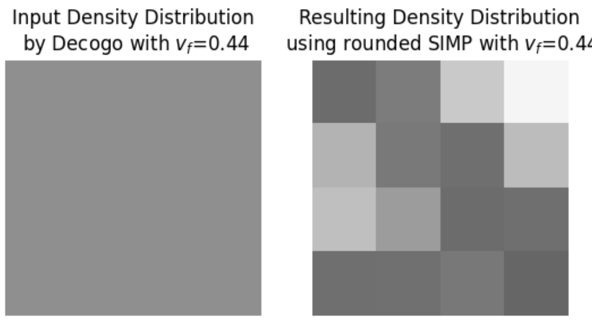


Figure 18: Resulting density distributions

In the next section, the reformulated TO problem is solved considering true mixed-integer variables, before an extension is shown that can handle the challenge of the static behavior.

6.6.2 MINLP

After the TO problem is solved using continuous variables to map the densities (NLP) with the new approach, this problem is attempted to be solved as a mixed-integer quadratic non-convex problems, which corresponds to the general problem (2.1). This is more difficult and requires minor adjustments in the process of solving the sub-problems, finding an initial feasible solution to eliminate the slacks and solving the original problems using an initial point provided by Decogo. As long as there is no global solver for sub-problems, the SIMP approach is used to imitate a true binary solution regarding the density variables.
For the sub-problems the solutions obtained from the SIMP approach, are simply rounded to integral numbers. After rounding, the corresponding displacements and compliance is calculated again. The following algorithm roughly describes this process.

Algorithm 8 Solve Sub-Problem as MINLP

---

function SOLVESUBMIP($x_{k,init}$, d_k)
    $g_k \leftarrow GetBoundaryForces(x_{k,init})$
    $\rho_k, u_k, c_k \leftarrow SIMP(x_{k,init}, d_k, g_k)$
    $\tilde{\rho}_k \leftarrow Round(\rho_k)$
    $\tilde{u}, \tilde{c} \leftarrow TransformRelaxatedPoint(\tilde{\rho})$
    return $\tilde{\rho}, \tilde{u}, \tilde{c}$
end function

The function `TransformRelaxatedPoint` calculates the new relaxated values for the displacements and densities. For finding an initial solution candidate for the slack elimination, a strong heuristic is used, which should be avoided in further investigations. The SIMP approach performs one or two iterations on the original problem, then the densities are rounded to a integral number and the corresponding relaxated values are calculated again, like it was done for the sub-problems. The biggest problem of this process is that the relaxated densities lead to a violation of the volume constraint in many cases so that the point is not feasible regarding the global constraints and the slacks will not be eliminated. To avoid this issue, the right-hand side of the volume constraint, i.e. the target volume, is changed so that the relaxated point is feasible. This is a very strong heuristic, as the original problem is changed.

It is necessary to keep this in mind while comparing the results of the original problem and the results from this new approach. Further research should deal with removing this heuristic. Reasons for using this heuristic are the scale and simplicity of this report. As it is defined by the CG method, this process does not restrict the global optimality. A second issue of this process is that simply rounding the densities while eliminating the slack variables can lead to unreasonable geometries so that there are very major displacements due to unconnected elements. For the experimental model, this issue was avoided by varying the number of SIMP iterations. The number was determined just by testing different numbers of main iterations, which has shown that two iterations lead to usable results. This number strongly depends on the model. To map the MINLP problem in the implementation, the configuration key `MINLP` is set to `True`, while the rest of the configuration stays the same as used for the NLP problem (6.6.1). The results of solving the TO problem as a MINLP problem are shown in the table below.

Model statistics		
Number of blocks/ sub-domains		4
Total number of variables		88
	Density variables	16
	Displacement variables	72
Variable numbers in blocks		
	Sub-domain 1	22
	Sub-domain 2	22
	Sub-domain 3	22
	Sub-domain 4	22
Number of global constraint		23
Solver statistics		
Total time [seconds]		50.60
Main iterations		5
Number of CG iterations		26
Number of solved sub-problems		148
Number of generated columns		10
Sum of slacks in MP		0.96
Maximum slack value in MP		0.96
Results		
Primal bound of RTO		13.59
CG relaxation objective value		24.26
Quality gap		43.98%

Table 8: Results for experimental model solved by Decogo as MINLP problem

Compared to the table of the NLP results (table 7), some parallels can be pointed out. The solver neither able to improve the objective value of 24.26

of the master problem nor to reduce the duality gap. The reason for this should be the same, as only ten columns were generated and added to the master problem. In addition there are still slacks, although a feasible point is presented to the MP. The small values for the objectives of the primal bound and the master problem stand out immediately, as they are significantly smaller than if the experimental model is solved with the classic SIMP approach without methods of column generation. This can be explained by the rounding: the rounded solutions of the original problem and the block-separated problem are the same, so this is no improvement compared to solving a domain with the SIMP approach directly and strongly depends on the rounding heuristic. The number of solved sub-problems is 148 and significantly deviates from the first variant. It seems that the problem is more flexible, but there are also only ten columns added, so this is not a real improvement. Furthermore, the analysis of the input density distribution (figure 19) shows that the problem is static again, because it is the same as presented as feasible point to the master problem.

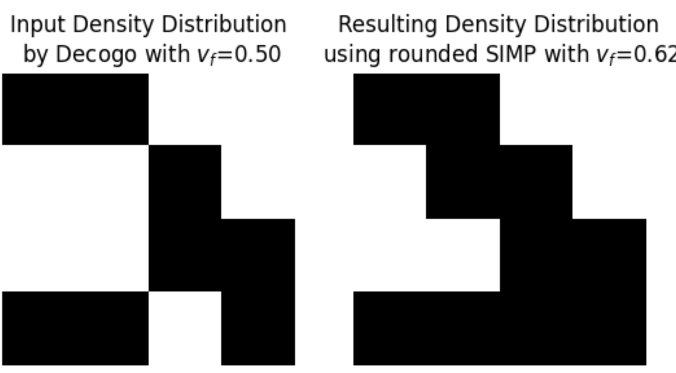


Figure 19: Resulting density distributions of MINLP TO problem

Solving both variants with the basic new approach shows that Decogo can understand a TO problem but is not able to solve it leading to good results yet. There are some extensions which are used to manage the challenge of not finding better columns. These extensions are presented in the next section.

6.7 Extensions

This section presents some prepared extensions, which can lead to an improvement of the new approach and handle the main issue of not reducing the dual gap between the primal bound and the objective function's value of the master problems. Due to the scale of this report, the presented extensions are only rudimentarily implemented in the current TO layer at this point. Some of them also require changing the source code while using the required testing. Furthermore, they are not fully tested regarding usage and their influence on the results. They are only intended to provide starting points for further research. Nevertheless, the current implementation contains some configurable functions to map some of these extensions.

6.7.1 Pertubation Extension

The first results for both variants show that Decogo struggles to find better columns which can improve the objective value of the master problem. Due to this issue, the duality gap between the master problem and the original can not be decreased. As the boundary forces can vary, but this is not mapped while solving sub-problems, it seemed that the master problem is not flexible regarding those boundary forces. It is expected that the most sub-problems are similar to each other regarding the density distribution, which means that an element with a high density after solving the sub-problem always has a high density. The same applies to elements with low densities. For elements with a medium density, it can vary. The thresholds for defining high and low densities are user-specific. In addition to of varying the density distribution, the boundary force vector $\vec{g}_k$ is varied to make the master problem more flexible in terms of the boundary forces so that the CG methods may find new columns. Varying the boundary forces depends on a factor which determines a range for the new pertubated force vector. The following algorithm outlines the creation of a pertubated feasible point related to a solution point of a sub-problem. For this the generated point needs to fulfill the local constraints X_k.

Algorithm 9 Create pertubated local point

> **function** PERTUBATEPOINT($\underline{\rho}_k^*$, $\vec{f}_k$)
> $\underline{\rho}_{ks} \leftarrow \underline{\rho}_k^*$
>   **for** $\rho_e \in \underline{\rho}_k^*$ **do**
>   **if** $\rho_e > \epsilon^+$ **then**
> $\rho_e \leftarrow 1$
>   **else if** $\rho_e < \epsilon^-$ **then**
> $\rho_e \leftarrow 0$
>   **else**
> $\rho_e \leftarrow R_\rho, \;\; R_\rho \in \{0, 1\}$
>   **end if**
>   **end for**
> $\vec{f}_{kp} \leftarrow \vec{f}_k \cdot R_f, \;\; R_f \in \left[1 - \frac{r_f}{2}, 1 + \frac{r_f}{2}\right], \;\; r_f \in [0, 1]$
> $\vec{u}_{kp} \leftarrow FEA(\rho_{ks}, \vec{f}_{kp})$
>   **return** $\underline{\rho}_{ks}$, $\vec{u}_{kp}$
> **end function**

ϵ^- and ϵ^+ denote the lower and upper threshold for determining low and high densities. The available range for the force vector is denoted by r_f. R indicates a random number.
For the implementation, the parameters of this extension, i.e. the thresholds and force vector range, can be configured by a dictionary, which is part of the total configuration for the TO layer. Below, an example of this sub-dictionary is given

```
'Pertubation': {
        'adding_points': 10,
        'threshold': (0.3, 0.7),  # (lower upper)
        'force_range': 0.8
}
```

The key `adding_points` determines how many similar points are created after solving one sub-problem.

First tests with this extension by applying it to the NLP variant of the TO problem show that as expected, a lot of feasible columns can be generated and added to the master problem. Due to that adding of columns, the solver is able to improve the objective value of the master problem, as better columns are provided to the master problem by this pertubation extension. Figure 20 shows this development during the optimization process since the slack variables are eliminated for the NLP variant.

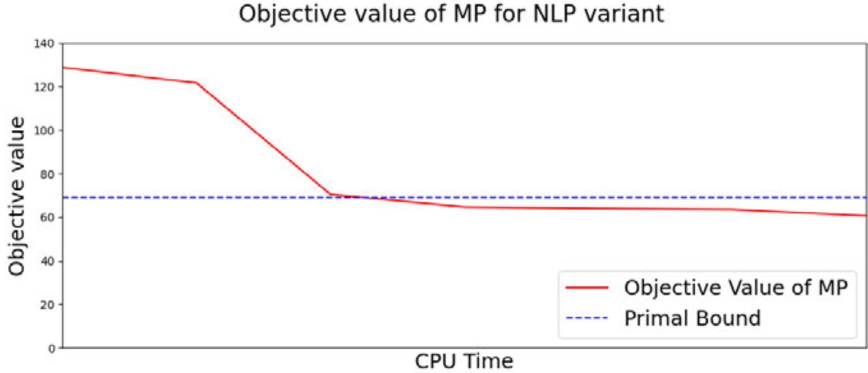


Figure 20: Objective value of MP for NLP variant

The objective value is reduced from 128.781 to 60.589, which in a reasonable range with view to the primal bound and leads to a resulting duality gap of 12.39%. For these results a total number 2778 columns were added to the master problem. In addition, the primal bound equals 69.157 which is an improvement of 0.14% compared to the result for directly solving the experimental model with the SIMP approach. This is a very minor improvement, but can be an indicator that the new approach leads to the desired results.

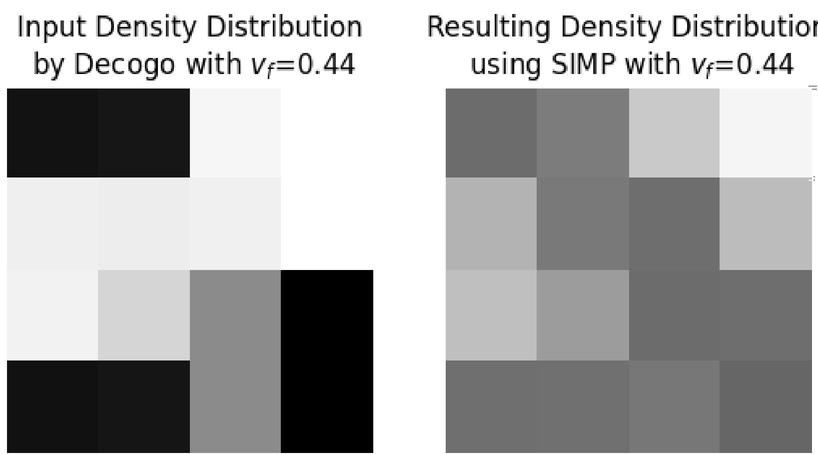


Figure 21: Density distributions for column pertubation

As shown, the density distribution provided by the master problem has some similar characteristics compared to the SIMP solution. The elements in the upper right corner have zero or a very small density so they can be omitted. The same applies for several elements on the left side of the model. There are also some elements with high densities that match to the expected solution. The provided density distribution does not fit perfectly but it is a basis for further research and improvements of this new approach. It seems that the target volume restricts the problems strongly, as it is fully exhausted.
Taking the MINLP variant including the pertubation extension into consideration proves that the algorithm was able to eliminate the slack variables, by providing a pertubated inner point. For this quick test, 1286 inner points were added to the master problem. The objective value of the master problem equals to the primal bound so the quality gap is fully closed. On a first glance, this seems to be reasonable, but the details reveal that the problem is still very static, which can easily be explained: This striking feature are coursed by the presented feasible point to the master problem for trying to eliminate the slacks, as the compliance of that point corresponds to

the objective value of the master problem. This points depends on that strong heuristic of rounding the solution of the SIMP approach. In addition, the objective value is constant during the optimization process. The static character is also outlined by considering the input density distribution provided by the master problem (figure 22). In addition, it is the same as without applying the pertubation extension. The solution of the rounded SIMP approach is not considered in detail because the volume constraint is violated. It can only be derived that the volume constraint strongly restricts an optimal solution.

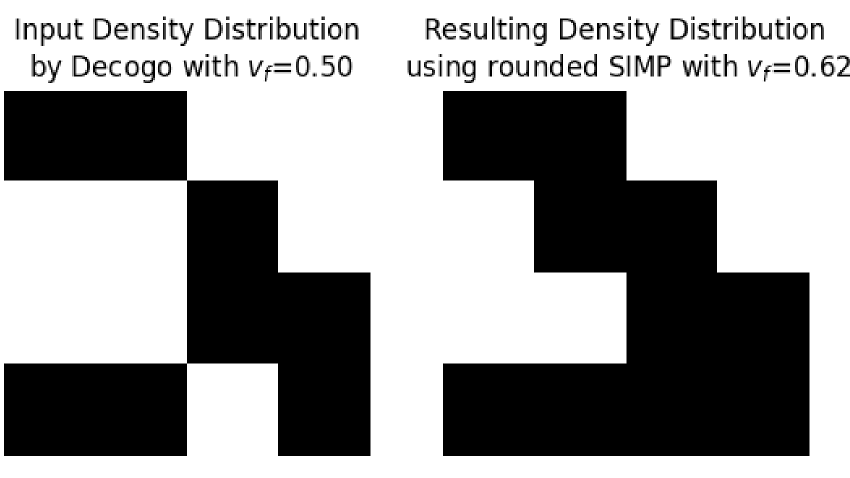


Figure 22: Density distributions for MINLP variant with pertubation extension

Nevertheless, the master problem approximates the original problem. This confirms the assumption of the feasibility of the new approach, as it is possible to generate a linear master problem which maps the original TO problem regarding its bounds and constraints. As long as a global solver for small topology optimization problems is not available, the objective value of the master problem can not be improved further.

By showing that it is possible to improve the objective value of the master problem by the solutions of the generated sub-problems, and by showing that the linear master problem can approximate a TO problem in terms of its bounds and constraints, it is proposed to continue working on this new generate-and-solve approach to tackle the elaborated challenges. Moreover, this approach still has the potential to determine a global solution to the general TO problem. In addition, this approach provides the possibility of parallel solving sub-problems.

6.7.2 Direction Stabilizer Extension

There is a theory for reducing the required iterations while solving a large optimization problem using the column generation algorithm. This method is outlined in the paper "Stabilized column generation" by O. du Merle [du +99] and could be interesting for this approach for further research. The basic functionality is still implemented but not fully tested. Just using the configuration to apply this method does not lead to improvements of the results regarding neither the performance nor the duality gap. The configuration is mapped with a dictionary, like it is done for the pertubation extension, in the main configuration dictionary for the TO layer.

7 Conclusion

This thesis has given a proof of concept of a new decomposition-based approach for solving topology optimization problems using methods of column generation and has determined its usefulness and feasibility. Generally, the quadratic non-convex TO problem is solved by reformulating the general TO problem to a block-separable problem (2.2) and alternately solving a linear master problem (inner approximation of convex hull) and corresponding sub-problems. The sub-problems of the TO problem are generated (5.3) by applying domain decomposition to the original problem (3.8). Solutions of these small sub-problems are used to update the master problem to improve the quality of the approximation. After the new approach converges (reduced costs are greater or equal than zero), the master problem provides a true lower bound of the original problem. In addition, it provides an estimated optimal solution of the original problem and a search direction that can be used to determine the global optimal solution of the original problem. The basis of this new approach is the column generation algorithm which is already applied to large optimization problems. The real innovation of this approach is solving TO problems with the help of a dual master problem by using sub-problems in accordance with the original problem and determining a true lower bound of the original TO problem. Hopefully, this method can improve the quality of conventional approaches for solving TO problems, or rather find better solutions in general, as most of them solve a TO problem only locally so that it is not known whether better solutions exist. Furthermore, this approach provides the possibility for a parallel calculation. In addition, it is easy to use any conventional approach for this new algorithm for solving sub-problems, as the new algorithm is generic. Since the proof of concept was the aim of the work, the experimental model (3.7) was deliberately chosen to be simple in order to obtain initial results for the new approach with as little effort as possible. Further research should deal with the scalability of the TO models, so that application-oriented structures can be optimized. There are basically two different views on the new approach: the theoretical view and the implementation view.
The complete implementation can be found in a "GitHub" repository located here: "aWsKlixz/DecogoTOLayer" [Tol]. It also provide the examples presented in this report including the experimental model. There are some further features that are not necessarily required for this new approach. In terms of functionality, the implemented layer for Decogo can understand a TO problem and apply the CG methods to it. Topology optimization models, or rather their finite elements models, can be created programmatically or by using the open-source tool "GMSH". In addition, the implementation

contains a rewritten SIMP solver and a prepared framework for arbitrary meshes with more complex element formulations than the rectangular two-dimensional elements. Moreover, there are some functionalities to visualize the solving progress, TO domains and printing information about the current TO problem. Some of these functionalities are briefly presented in section 9. The project's size for the implementation became large, so it may be required to reorganize it to improve the usage of the different functionalities. Furthermore, the current implementation only contains the SIMP solver to calculate the results of a TO problem. The source code is not commented everywhere, which might lead to confusion. If there is more time and this project is continued, there will be some aspects to improve the performance and usage of the current implementation. Some of those aspects are presented in the next section.
As to be expected, there were a number of challenges in the development of the theory and comparing them to the received results from this new approach. Originally, it was intended that this new approach does not require any heuristics so that the problem is definitely solved globally. However, heuristics are still needed to receive solutions of static undetermined sub-problems (3.5.3) by solving a least-squares problem instead of exactly calculating the required displacements directly (3.5.2). During this process, a trust region for the displacements is generated and it is not fully clarified whether this region can restrict the problem in such a way that it can not be solved globally anymore. Currently, a hard trust region is defined in the implementation. Another idea would be a flexible trust region by updating it each time a static undetermined sub-problem is solved. Both variants of the trust region require further research to determine their impact on the global optimality. The second heuristic is used while eliminating the slack variables of the master problem of the MINLP variant, by changing the original by modifying the right-hand side of the volume constraint. This was done to eliminate the slack variables in the first place, so the CG algorithms can find feasible points for the master problem. It was also done because of the scale of this report. Analyzing the results generated with the new approach including the intermediate solutions of the sub-problems leads to the assumption that the SIMP approach is unsuitable for the new approach as it blurs good starting points provided by the master problem to a more fractional density distribution. This may happen due to the applied filter or due to the OC approach in general. Further research are required to determine whether the SIMP approach could be at least used to determine approximate solutions of sub-problems, or rather can be used as the required local solver. Another challenge is improving the master problem's objective value and simultaneously reducing the duality gap to determine the global optimality of this

new approach. As long as only the SIMP solver exists, this will not succeed because the SIMP approach has a heuristic character, so it is only a local solver regarding the original TO problem, but the master problem is solved globally.

Despite the revealed challenges and issues outlined above, this report confirms the feasibility and usefulness of the new generate-and-solve approach for solving TO problems using column generation methods. It is possible to generate a linear master problem, which is easy to solve and similar to an original quadratic non-convex topology optimization problem, using a block-separable reformulation of a TO problem. The calculations have shown that the column generation algorithm was able to improve the objective value of the master problem only by solving sub-problems of the TO problem. A very interesting discovery is that the resulting objective value of the inner approximation almost matches the compliance of the expected optimal solution of the topology optimization problem. This observation underpins the assumption that the original TO problem and the master problem span a relaxated convex hull; however this needs to be proved. At this point in time, the implemented solver determines a local solution using the new approach because it contains heuristics but nevertheless, this method has the potential to solve a TO problem globally. Regarding all of those heuristics, some suggestions for further research exist with the aim of avoiding or crossing them out. Due to the scale of this report, the implementation is not fully elaborated regarding the global optimality. In the following section, the suggestions for avoiding the heuristics, for improving the performance of the implementations and reaching the global optimality are outlined.

8 Prospect

While developing the new decomposition-based approach for TO problems and creating the implementation for the TO layer of Decogo, some suggestions have been worked out to improve usage, performance and quality and to avoid the outlined heuristics. These extensions would exceed the scale of this report as during this report, only a prototype for the new approach was developed to determine the feasibility and usefulness. In this section, the suggestions related to the revealed challenges for further research are summarized. As this approach is generic and Decogo is still under development, improvements of Decogo will lead to improvements of this new approach. This section starts by presenting possible solutions to reduce the duality gap while the master problem's objective value is improved, so that the quality of the approximation is improved, leading to an estimated optimal solution closer to the global optimal solution. In addition, some ideas are outlined to improve solving sub-problems. After that, an unproven idea is given to remove the volume constraint from the general TO problem to avoid the strong simplification while trying to find a feasible solution candidate to remove the slacks from the master problem. Lastly, some suggestions and ideas are presented related to the implementation.

8.1 Reaching Global Optimality

The new approach has the potential to solve the TO problem globally. To reach this quality of the results, it is necessary to reduce the duality gap so that the final estimated solution point, provided by the master problem, is close to the global solution. One advantage of the new approach is using solutions of sub-problems to estimate the final solution. If these sub-problems are solved globally, it will be possible to reduce the quality gap more an provide a reasonable estimated starting point to find the global solution of the original problem. As the sub-problems are small compared to the original problem, it is possible to develop a global solver for these small problems, e.g. using the "SCIP Optimization Suite" [Bes+21]. For a beam framework, this was already done in the master project "Global Optimization of Beam Structures". The corresponding implementation is found in the following "GitHub" repository "aWsKlixz/PyomoBeamOptim" [Pyb]. Using a global solver will show that the primal bound of the original problem converges to the global solution. The optimization process will perform as follows: Firstly, the columns will be initialized using that sub-gradient method of "Decogo" to find initial columns for the master problem. Then, the slacks

will be eliminated by approximated columns made of local solutions of the sub-problems by presenting a feasible point of the original problem to the master problem. After that, global solutions of the sub-problems are used in the third phase of Decogo for the column generation algorithm to find better columns and to further improve the objective of the master problem. If no new columns are found by the global solver in the third phase, the quality of the approximation due to the master problem is already good due to the local solutions of the sub-problems. If the estimated value of the master problem is close to the global solution, it could be interesting how conventional approaches perform using that estimated solution as a start point for searching the global solution of the original problem.

8.2 Locally Solving Sub-Problems

In section 4, different approaches are presented to solve TO problems. These approaches also contain heuristics but they may perform better than the SIMP approach to solve the sub-problems locally, like evolutionary TOSS approaches. They may lead to a solution which better matches to the global solution, e.g. by having a more binary character, so the total number of required columns can be reduced. Furthermore, as artificial intelligence and machine learning methods become more popular and perform better regarding calculation time and quality, they can be interesting for further research. A starting point could be the paper "3D Topology Optimization Using Convolutional Neural Networks" [Ban+18].

8.3 Impact of Volume Constraint

The general and the corresponding block-separated TO problem described in section 2 contains a volume constraint. During the development process in this report, this constraint has led to issues repeatedly. It is the main course for the fact that the SIMP approach depends on an initial value and prevents the solver from finding a true binary solution, as many of that true binary solutions would violate this constraint (6.6.2). From the user's point of view, the percentage target volume v_f needs to be predefined, but it is practically impossible to define a logical value for it related to a complex application-oriented structure. Due to these issues, further research should address the elimination or replacement of this constraint. Regarding the SIMP approach, the independence of v_f can be reached by a multi-resolution approach presented in the paper "A computational paradigm for multiresolution topology optimization (MTOP)" [Ngu+10]. This approach was roughly tested on the experimental model and it seems that the desired results are achieved. Due

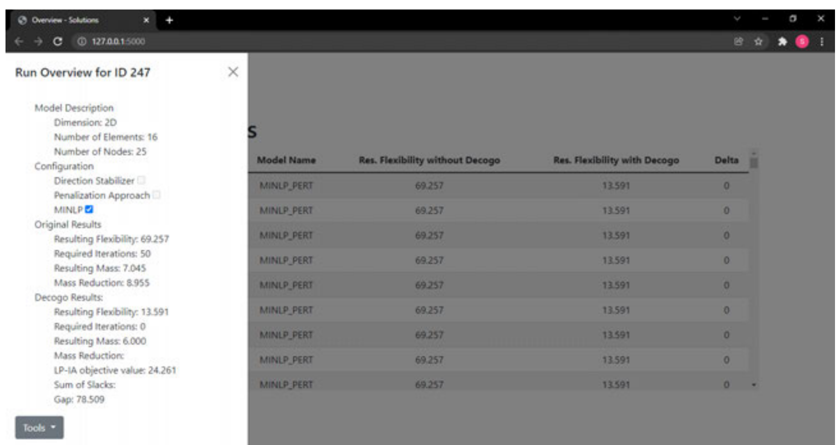
to the scale of this report, this approach has not been pursued further. The main disadvantage of this method is that the original problem is changed, like it was done for the slack elimination for the MINLP variant (6.6.2). Another idea to avoid the volume constraint is using a maximum allowed stress constraint directly, so that the displacements are restricted indirectly. This could also be used for solving static undetermined elasticity problems of the sub-domains. But this may require fundamental reformulation of the general TO problem 2.1.

8.4 Implementation

Suggestions for the implementation also exist to improve the performance and handling. The sparse form from the `scipy.sparse` module should be implemented for the FEA to be able to handle more complex geometries in an appropriate time. Regarding application-oriented structures, the framework of the finite elements needs to be continued by adding further element formulations. As it could make sense to test different solvers for the subproblems, a solver framework could help to quickly replace these sub-solvers in a modular way.

9 Additional and Prepared Features

At the end of this report, this section briefly presents some additional and prepared features of the current implementation.
Since the pertubation extension (6.7.1) was added to the new approach, the idea came up to save calculations runs as they depend on random values. For this, a simple SQLight database was created. It stores general data on a calculation run in columns. The complete information including the model's parameter, the run's configuration, the results using a conventional approach and the results using the new approach, are stored in a separated column as a `JSON` string. This string can easily be interpreted as a Python dictionary. To view the results of a specific run, a basic web application based on the `flask` framework [Fla] was created. A screenshot is provided below. In addition, the implementation contains a reader to translate the JSON string back to model objects, so that they can be visualized again.



The implementation contains a folder called "tests", which provides some first unittests. The current implemented tests can be used to test whether an original TO problem is reformulated correctly, e.g. both formulations lead to the same objective value for the same input parameters. Furthermore, the test reviews whether the copy constraints are fulfilled for the initial state of the model depending on the initial solutions of the FEA.
The figures, which present stages of the experimental model in this report, were created by a class called `Visualizer`, which can be found in "pylib/visualizer/visualizer.py". It creates those figures based on a `TOModelBase` object and the corresponding decomposer. These presented features are not perfect but work for the current state of the project and will be continued in further research.

References

[ARE08] Makrizi Abdelilah, Bouchaib Radi, and Abdelkhalak ELHami. "Solution of the Topology Optimization Problem Based Subdomains Method". In: *Applied Mathematical Sciences* 2 (Jan. 2008), pp. 2029–2045.

[Ban+18] Saurabh Banga et al. "3d topology optimization using convolutional neural networks". In: *arXiv preprint arXiv:1808.07440* (2018).

[Bat16] K.J. Bathe. *Finite Element Procedures*. Prentice Hall, 2016. ISBN: 9780979004957. URL: `https://books.google.de/books?id=rWvefGICfO8C`.

[Bes+21] Ksenia Bestuzheva et al. *The SCIP Optimization Suite 8.0*. Technical Report. Optimization Online, Dec. 2021. URL: `\texttt{http://www.optimization-online.org/DB_HTML/2021/12/8728.html}`.

[Byn+21] Michael L. Bynum et al. *Pyomo–optimization modeling in python*. Third. Vol. 67. Springer Science & Business Media, 2021.

[du +99] Olivier du Merle et al. "Stabilized column generation". In: *Discrete Mathematics* 194.1 (1999), pp. 229–237. ISSN: 0012-365X. DOI: `https://doi.org/10.1016/S0012-365X(98)00213-1`. URL: `https://www.sciencedirect.com/science/article/pii/S0012365X98002131`.

[Fla] *Flask Web Framework*. The Pallets Projects. Jan. 16, 2022. URL: `https://flask.palletsprojects.com/en/2.0.x/` (visited on 01/16/2022).

[GR09] Christophe Geuzaine and Jean-François Remacle. "Gmsh : a three-dimensional finite element mesh generator with built-in pre-and post-processing facilities". In: 2009.

[Gur] *Gurobi Solver*. URL: `https://www.gurobi.com/`.

[Hun+17] William Hunter et al. *ToPy - Topology optimization with Python*. 2017. URL: `https://github.com/williamhunter/topy`.

[Mlc] *Column Generation, Dantzig-Wolfe, Branch-Price-and-Cut*. URL: `\texttt{https://co-at-work.zib.de/slides/Donnerstag_24.9/cgbpdw-coatwork-annotated.pdf}`.

[Mut+21] Pavlo Muts et al. "A column generation algorithm for solving energy system planning problems". In: *Optimization and Engineering* (2021), pp. 1–35.

[Ngu+10] Tam H Nguyen et al. "A computational paradigm for multiresolution topology optimization (MTOP)". In: *Structural and Multidisciplinary Optimization* 41.4 (2010), pp. 525–539.

[Nik04] GP Nikishkov. "Introduction to the finite element method". In: *University of Aizu* (2004), pp. 1–70.

[Num] *numpy.linalg.solve*. URL: `https://numpy.org/doc/stable/reference/generated/numpy.linalg.solve.html`.

[Pyb] *PyomoBeamOptim*. URL: `\texttt{https://github.com/aWsKlixz/PyomoBeamOptim}`.

[Sci] *`scipy.optmize.least_squares`*. URL: `\texttt{https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html#scipy.optimize.least_squares}`.

[Sig] O Sigmund. *Structural and Multidisciplinary Optimization. A 99 line topology optimization code written in Matlab*. English. JOUR. Structural and Multidisciplinary Optimization: Springer. URL: `https://doi.org/10.1007/s001580050176`.

[SS13] Krister Svanberg and H Svard. "Density filters for topology optimization based on the geometric and harmonic means". In: *10th world congress on structural and multidisciplinary optimization. Orlando*. 2013.

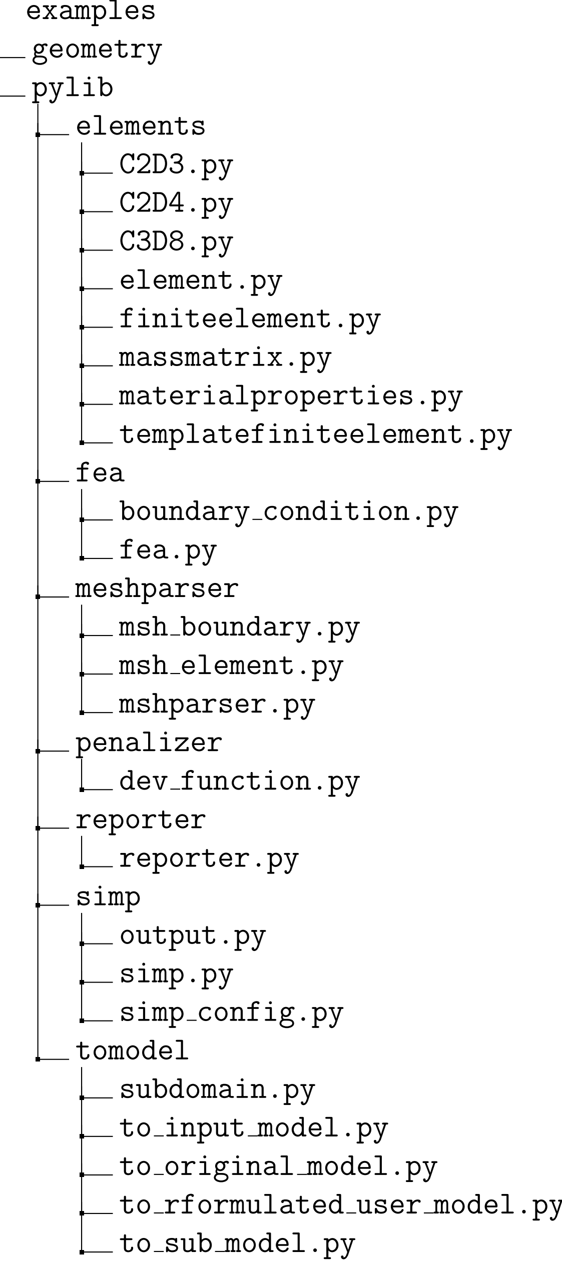
[Tol] *DecogoTOLayer*. URL: `\texttt{https://github.com/aWsKlixz/DecogoTOLayer}`.

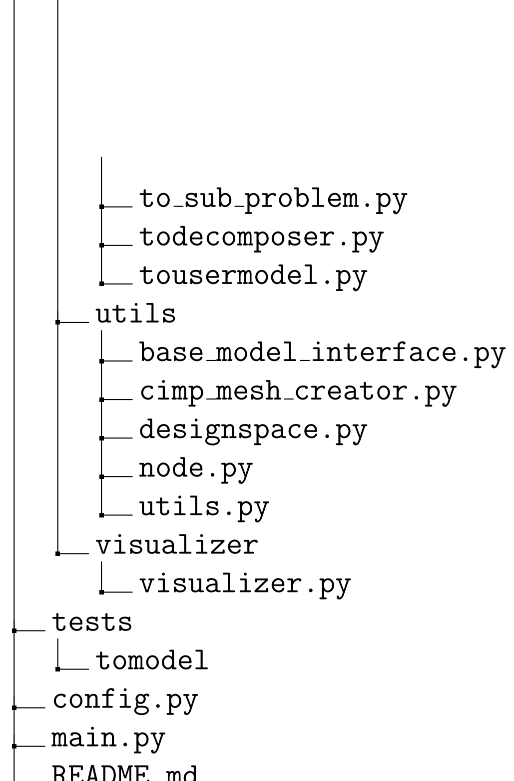
[Z88] *Theory User Manual*. URL: `http://download.z88.de/z88arion/V2/benutzerhandbuch.pdf`.

A Annex

A.1 Folder Structure

This section provides the folder structure for the topology optimization layer for Decogo. The highlighted structure does not include the web application and temporary testing files. The folder 'geometry' contains different mesh files.




to_sub_problem.py
todecomposer.py
tousermodel.py
utils
base_model_interface.py
cimp_mesh_creator.py
designspace.py
node.py
utils.py
visualizer
visualizer.py
tests
tomodel
config.py
main.py
README.md

A.2 Programmatic Model

The example below shows how to create the experimental model in a programmatic way. At the end of this script, an object of the `TOModelBase` class is provided.

```
import numpy as np

from pylib.elements.element import Element
from pylib.elements.C2D4Simple import ke
from pylib.utils.node import Node

from pylib.tomodel.tousermodel import TOModelBase
from pylib.tomodel.boundary_condition import BoundaryConditions

nodes = {}      # initialize dict for nodes
elements = {}   # initialize dict for elements

base_nodes = list(range(1, 26))         # set up the list of node-ids; N =
    {1,...,25}
element_numbers = list(range(1, 17))    # set up list of element-ids; M =
    {1,...,16}

# Design Space Section; works only for this model because of the 2
    dimensional space and square elements
element_space = np.array(element_numbers).reshape((4, 4)).T     # set up a
    2D design space for the elements
node_space = np.array(base_nodes).reshape((5, 5)).T             # set up a
    2D design space for the nodes

col = -1    # will be equivalent to x-coordinate of a node
row = 0     # will be equivalent to y-coordinate of a node
# Determine dict for nodes
for i, n_id in enumerate(base_nodes):
    if n_id % 5 == 1:
        row = 4
        col += 1
    nodes[n_id] = Node(n_id, float(col), float(row), z=0.0, index=i)    #
        Node object is created
    row -= 1
# Determine dict for elements
for e_id in element_numbers:
    i = int(np.where(element_space == e_id)[0])     # get x-position in
        design space
    j = int(np.where(element_space == e_id)[1])     # get y-position in
        design space

    element_nodes = node_space[i:i+2, j:j+2].reshape((4, ))     # get the
        corresponding nodes
    element_nodes = element_nodes[[2, 3, 1, 0]]                 # reorder
        node-ids so they fit to element formulation
    elements[e_id] = Element(e_id, {n: nodes[n] for n in element_nodes},
        ke(1, 0.3), dense_x=0.44)  # Element object is created

outer_forced_nodes = [25]                                       # create a
    list of loaded nodes
f_dict = {n: np.array([0, -1]) for n in outer_forced_nodes}     # create
    the dict for outer forces
fixed_nodes = [1, 5]                                            # create a
    list of fixed nodes
```

```
bc = BoundaryConditions(f_dict, fixed_nodes, list(base_nodes))  # create
    the boundary conditions nodes

base_model = TOModelBase(nodes, elements, bc, 2)                # create
    the instance of TOModelBase

# base_model can be used for further actions or rather will be imported for
    optimization
```

A.3 Mesh File

Below, the .msh-file for the experimental model is provided.

```
$MeshFormat
2.2 0 8
$EndMeshFormat
$PhysicalNames
3
1 1 "fixation"
1 2 "load"
2 5 "plane"
$EndPhysicalNames
$Nodes
25
1 0 0 0
2 4 0 0
3 4 4 0
4 0 4 0
5 1 0 0
6 2 0 0
7 3 0 0
8 4 1 0
9 4 2 0
10 4 3 0
11 3 4 0
12 2 4 0
13 1 4 0
14 0 3 0
15 0 2 0
16 0 1 0
17 1 1 0
18 1 2 0
19 1 3 0
20 2 1 0
21 2 2 0
22 2 3 0
23 3 1 0
24 3 2 0
25 3 3 0
$EndNodes
$Elements
```

```
28
1 15 2 0 1 1
2 15 2 0 2 2
3 15 2 0 3 3
4 15 2 0 4 4
5 1 2 0 1 1 16
6 1 2 0 1 16 15
7 1 2 0 1 15 14
8 1 2 0 1 14 4
9 1 2 0 2 2 8
10 1 2 0 2 8 9
11 1 2 0 2 9 10
12 1 2 0 2 10 3
13 3 2 0 1 1 5 17 16
14 3 2 0 1 16 17 18 15
15 3 2 0 1 15 18 19 14
16 3 2 0 1 14 19 13 4
17 3 2 0 1 5 6 20 17
18 3 2 0 1 17 20 21 18
19 3 2 0 1 18 21 22 19
20 3 2 0 1 19 22 12 13
21 3 2 0 1 6 7 23 20
22 3 2 0 1 20 23 24 21
23 3 2 0 1 21 24 25 22
24 3 2 0 1 22 25 11 12
25 3 2 0 1 7 2 8 23
26 3 2 0 1 23 8 9 24
27 3 2 0 1 24 9 10 25
28 3 2 0 1 25 10 3 11
$EndElements
```

A.4 Configuration Dictionary

Here, an example of a configuration dictionary for the TO layer is presented. It also contains some prepared extensions which are not part of this report.

```
Configuration_TOLayer = {
    'use_stabilizer': False,
    'use_penalization_approach': False,
    'MINLP': False,
    'is_reported': True,
    'report_name': 'Complex',

    'SimpProblem': {
        'volfrac': 0.44,
        'penalization_factor': 3,
        'emin': 1e-9,
        'emax': 1,
        'rmin': 1.44,
        'filter': 'sensitivity'
    },

    'FEModel': {
        'preferred_solver': 'direct'
    },

    'Pertubation': {
        'adding_points': 10,
        'fast_sol_simp_iter': 2,
        'threshold': (0.3, 0.7),  # (lower upper)
        'force_range': 0.4
    },

    'CGPenalization': {
        'factor': 2.0
    },
    'CGStabilizer': {
        'delta': 1.0,
        'eps': 1.0
    }

}
```

A.5 Log Files

The presented log files in this section were used to visualize the results. In the log files, the density distributions that were presented to the sub-problems are also highlighted.

A.5.1 NLP Variant

```
Block separable reformulation:
Number of blocks:                                4
Number of nonlinear blocks:                      0
Min size of blocks:                              22
Max size of blocks (without linear blocks):      22
Max size of blocks (including linear blocks):    22
Number of vars:                                  88
Number of global constraints:                    23
Number of nonzero resources per block:           15,12,11,11
Number of equal./inequal. of global constraints:  22/1
-----------------------------------------------------
Used time: inf
-----------------------------------------------------
Initialization
Subgradient steps
Subgra.iter    Lagrange bound                alpha
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
1              84.86239791836496             1
[0.62094513 0.27034214 0.53394013 0.3348674 ]
[0.38414203 0.36013004 0.54796867 0.46747253]
[0.56272047 0.43976428 0.46721342 0.28937013]
[0.28886948 0.59156963 0.38188732 0.4963061 ]
2              33213.80698719126             1
[0.62094513 0.27034214 0.53394013 0.3348674 ]
[0.38414203 0.36013004 0.54796867 0.46747253]
[0.56272047 0.43976428 0.46721342 0.28937013]
[0.28886948 0.59156963 0.38188732 0.4963061 ]
3              294986.8112530708             2
[0.62094513 0.27034214 0.53394013 0.3348674 ]
[0.38414203 0.36013004 0.54796867 0.46747253]
[0.56272047 0.43976428 0.46721342 0.28937013]
[0.28886948 0.59156963 0.38188732 0.4963061 ]
4              275914.3181959714             4
[0.62094513 0.27034214 0.53394013 0.3348674 ]
[0.38414203 0.36013004 0.54796867 0.46747253]
[0.56272047 0.43976428 0.46721342 0.28937013]
[0.28886948 0.59156963 0.38188732 0.4963061 ]
5              1527928.509851672             2.0
Time used for SubGradient: --9.67-- seconds
-----------------------------------------------------
Elapsed time: inf
-----------------------------------------------------
=====================================================
Column generation: approximated subproblem solving
Initial CG objective value: 97.20469586403783
CG iter        IA obj. value                 max slack value IA            sum slack values IA
1              97.20469586403783             28.231050743601976            268.2628376421817
[0.62094513 0.27034214 0.53394013 0.3348674 ]
[0.38414203 0.36013004 0.54796867 0.46747253]
[0.56272047 0.43976428 0.46721342 0.28937013]
[0.28886948 0.59156963 0.38188732 0.4963061 ]
[0.62094513 0.27034214 0.53394013 0.3348674 ]
[0.38414203 0.36013004 0.54796867 0.46747253]
[0.56272047 0.43976428 0.46721342 0.28937013]
[0.28886948 0.59156963 0.38188732 0.4963061 ]
Reduced costs greater than zero
New columns added: [0, 0, 0, 0]
number of minlp subproblems solved during CG: 0
=====================================================
Time used for init CG in iter 0: --7.51-- seconds
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
=====================================================
Find solution - init
Found approx. first primal bound of c=128.80
Pertubation Statistics: 0 of 10 points are infeasible
```

```
Time used for init FindSol in iter 0: --0.36-- seconds
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
Found the first feasible solution
IA obj. val: 97.20469586403783
Elapsed time: inf
---------------------------------------------------------
Fast column generation
iter      IA obj. value                 slacks
0         128.79669782543593            0.0
IA obj. val: 128.79669782543593
Elapsed time: inf
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
iter      IA obj. value                 slacks
1         128.79669782543593            0.0
IA obj. val: 128.79669782543593
Elapsed time: inf
New columns in FastCG:
[0, 0, 1, 0]
number of unfixed nlp subproblems solved during CG: 4
Time used for solving subproblem: --0.81-- seconds
---------------------------------------------------------
Time used for init cg fast fw in iter 1: --1.19-- seconds
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
---------------------------------------------------------
Fast column generation
iter      IA obj. value                 slacks
0         128.79669782543593            0.0
IA obj. val: 128.79669782543593
Elapsed time: inf
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
No new columns generated in the current iteration
New columns in FastCG:
[0, 0, 0, 0]
number of unfixed nlp subproblems solved during CG: 4
Time used for solving subproblem: --0.53-- seconds
---------------------------------------------------------
Time used for init cg fast fw in iter 2: --0.89-- seconds
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
---------------------------------------------------------
Fast column generation
iter      IA obj. value                 slacks
0         128.79669782543593            0.0
IA obj. val: 128.79669782543593
Elapsed time: inf
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
No new columns generated in the current iteration
New columns in FastCG:
[0, 0, 0, 0]
number of unfixed nlp subproblems solved during CG: 4
Time used for solving subproblem: --0.56-- seconds
---------------------------------------------------------
Time used for init cg fast fw in iter 3: --0.9-- seconds
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
---------------------------------------------------------
Fast column generation
iter      IA obj. value                 slacks
0         128.79669782543593            0.0
IA obj. val: 128.79669782543593
Elapsed time: inf
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
No new columns generated in the current iteration
New columns in FastCG:
[0, 0, 0, 0]
```

```
number of unfixed nlp subproblems solved during CG: 4
Time used for solving subproblem: --0.5-- seconds
---------------------------------------------------------
Time used for init cg fast fw in iter 4: --0.82-- seconds
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
---------------------------------------------------------
Fast column generation
iter      IA obj. value                 slacks
0         128.79669782543593            0.0
IA obj. val: 128.79669782543593
Elapsed time: inf
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
No new columns generated in the current iteration
New columns in FastCG:
[0, 0, 0, 0]
number of unfixed nlp subproblems solved during CG: 4
Time used for solving subproblem: --0.5-- seconds
---------------------------------------------------------
Time used for init cg fast fw in iter 5: --0.84-- seconds
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
CG relaxation obj. value in iter 0: 128.79669782543593
Time used for total init CG in iter 0: --12.5-- seconds
-----------------------------------------------------
Elapsed time at CG iter 0: --inf-- seconds
-----------------------------------------------------
=======================================================
Column generation
Initial CG objective value: 128.79669782543593
CG iter        IA obj. value                 max slack value IA            sum slack values IA
1              128.79669782543593            0.0                           0.0
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
[0.56272047 0.43976428 0.46721342 0.28937013]
[0.44 0.44 0.44 0.44]
Reduced costs greater than zero
New columns added: [0, 0, 0, 0]
number of minlp subproblems solved during CG: 4
=======================================================
CG relaxation obj. value in iter 1: 128.79669782543593
Time used for CG: --0.84-- seconds
-------------------------------------------------
Elapsed time at CG iter 1: --inf-- seconds
-------------------------------------------------
Num of MINLP subproblems solved in iter loop *1*   4
Total number of minlp subproblems solved in iter 1: 24
Total number of columns in iter 1: 10
Columns in blocks in iter 1: [2, 2, 4, 2]
Time used for CG in iter 1: --0.84-- seconds
------------------------------------
CG regarding all blocks
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
[0.56272047 0.43976428 0.46721342 0.28937013]
[0.44 0.44 0.44 0.44]
Time used for CG for all blocks: --0.77-- seconds
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
CG converges, checking the convergence by exact subproblem solving
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
[0.56272047 0.43976428 0.46721342 0.28937013]
[0.44 0.44 0.44 0.44]
=======================================================
Find solution - projection from ia solution - local search
Time used for FindSol in iter 1: --0.85-- seconds
-------------------------------------------------
Elapsed time at FindSol iter 1: --inf-- seconds
-------------------------------------------------
Total time used in iter 1: --3.03-- seconds
CG converged
Total time:                                      25.21238684654236
Reformulation time:                              0
Decomposition time:                              0
Containers time:                                 0
Primal bound:                                    68.963132875
```

```
Main iterations:                                 1
Number of CG iterations:                         2
CG relaxation obj. value:                        128.79669782543593
Number of MINLP subproblems:                     32
Number of unfixed NLP subproblems:               28
Number of fixed NLP subproblems:                 0
Number of solved sub-problems after CG:          60
Number of columns after CG:                      10
CG Gap (CG relaxation and primal bound):         46.4558186002
Total number of columns:                         10
```

A.5.2 MINLP Variant

```
Block separable reformulation:
Number of blocks:                                4
Number of nonlinear blocks:                      0
Min size of blocks:                              22
Max size of blocks (without linear blocks):      22
Max size of blocks (including linear blocks):    22
Number of vars:                                  88
Number of global constraints:                    23
Number of nonzero resources per block:           15,12,11,11
Number of equal./inequal. of global constraints: 22/1
-----------------------------------------------------
Used time: inf
-----------------------------------------------------
Initialization
Subgradient steps
Subgra.iter    Lagrange bound               alpha
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
1              84.86239791836496            1
[0.62094513 0.27034214 0.53394013 0.3348674 ]
[0.38414203 0.36013004 0.54796867 0.46747253]
[0.56272047 0.43976428 0.46721342 0.28937013]
[0.28886948 0.59156963 0.38188732 0.4963061 ]
2              33213.80698719126            1
[0.62094513 0.27034214 0.53394013 0.3348674 ]
[0.38414203 0.36013004 0.54796867 0.46747253]
[0.56272047 0.43976428 0.46721342 0.28937013]
[0.28886948 0.59156963 0.38188732 0.4963061 ]
3              294986.8112530708            2
[0.62094513 0.27034214 0.53394013 0.3348674 ]
[0.38414203 0.36013004 0.54796867 0.46747253]
[0.56272047 0.43976428 0.46721342 0.28937013]
[0.28886948 0.59156963 0.38188732 0.4963061 ]
4              275914.3181959714            4
[0.62094513 0.27034214 0.53394013 0.3348674 ]
[0.38414203 0.36013004 0.54796867 0.46747253]
[0.56272047 0.43976428 0.46721342 0.28937013]
[0.28886948 0.59156963 0.38188732 0.4963061 ]
5              1527928.509851672            2.0
Time used for SubGradient: --9.68-- seconds
-----------------------------------------------------
Elapsed time: inf
-----------------------------------------------------
=====================================================
Column generation: approximated subproblem solving
Initial CG objective value: 97.20469586403783
CG iter        IA obj. value                max slack value IA            sum slack values IA
1              97.20469586403783            28.231050743601976            268.2628376421817
[0.62094513 0.27034214 0.53394013 0.3348674 ]
[0.38414203 0.36013004 0.54796867 0.46747253]
[0.56272047 0.43976428 0.46721342 0.28937013]
[0.28886948 0.59156963 0.38188732 0.4963061 ]
[0.62094513 0.27034214 0.53394013 0.3348674 ]
[0.38414203 0.36013004 0.54796867 0.46747253]
[0.56272047 0.43976428 0.46721342 0.28937013]
[0.28886948 0.59156963 0.38188732 0.4963061 ]
Reduced costs greater than zero
New columns added: [0, 0, 0, 0]
number of minlp subproblems solved during CG: 0
=====================================================
Time used for init CG in iter 0: --8.13-- seconds
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
=====================================================
Find solution - init
```

```
Attention: Updated r.h.s. of volume constraint to 8.0 due to MINLP solving.
Found approx. first primal bound of MINLP with c=24.260951194847255
Found approx. first primal bound of c=24.26
Pertubation Statistics: 0 of 10 points are infeasible
Time used for init FindSol in iter 0: --0.41-- seconds
----------------------------------------------------
Elapsed time: --inf-- seconds
----------------------------------------------------
Found the first feasible solution
IA obj. val: 97.20469586403783
Elapsed time: inf
--------------------------------------------------------
Fast column generation
iter      IA obj. value                slacks
0         24.260951194847276           0.96
[1. 0. 1. 0.]
C:\FinalThesis\DecogoTOLayer\pylib\simp\simp.py:135: RuntimeWarning: invalid value encountered in true_divide
  np.minimum(1.0, np.minimum(x_old + move, x_old * (-dc / lmid) ** eta)))
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0. 1. 1. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0. 1. 0. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
No new columns generated in the current iteration
New columns in FastCG:
[0, 0, 0, 0]
number of unfixed nlp subproblems solved during CG: 4
Time used for solving subproblem: --0.11-- seconds
--------------------------------------------------------
Time used for init cg fast fw in iter 1: --0.42-- seconds
----------------------------------------------------
Elapsed time: --inf-- seconds
----------------------------------------------------
--------------------------------------------------------
Fast column generation
iter      IA obj. value                slacks
0         24.260951194847276           0.96
[1. 0. 1. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0. 1. 1. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0. 1. 0. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
No new columns generated in the current iteration
New columns in FastCG:
[0, 0, 0, 0]
number of unfixed nlp subproblems solved during CG: 4
Time used for solving subproblem: --0.12-- seconds
--------------------------------------------------------
Time used for init cg fast fw in iter 2: --0.43-- seconds
----------------------------------------------------
Elapsed time: --inf-- seconds
----------------------------------------------------
--------------------------------------------------------
Fast column generation
iter      IA obj. value                slacks
0         24.260951194847276           0.96
[1. 0. 1. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0. 1. 1. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0. 1. 0. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
No new columns generated in the current iteration
New columns in FastCG:
[0, 0, 0, 0]
number of unfixed nlp subproblems solved during CG: 4
Time used for solving subproblem: --0.13-- seconds
--------------------------------------------------------
Time used for init cg fast fw in iter 3: --0.46-- seconds
----------------------------------------------------
Elapsed time: --inf-- seconds
----------------------------------------------------
--------------------------------------------------------
Fast column generation
iter      IA obj. value                slacks
0         24.260951194847276           0.96
```

```
[1. 0. 1. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0. 1. 1. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0. 1. 0. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
No new columns generated in the current iteration
New columns in FastCG:
[0, 0, 0, 0]
number of unfixed nlp subproblems solved during CG: 4
Time used for solving subproblem: --0.11-- seconds
--------------------------------------------------------
Time used for init cg fast fw in iter 4: --0.42-- seconds
----------------------------------------------------
Elapsed time: --inf-- seconds
----------------------------------------------------
--------------------------------------------------------
Fast column generation
iter      IA obj. value                 slacks
0         24.260951194847276            0.96
[1. 0. 1. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0. 1. 1. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0. 1. 0. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
No new columns generated in the current iteration
New columns in FastCG:
[0, 0, 0, 0]
number of unfixed nlp subproblems solved during CG: 4
Time used for solving subproblem: --0.13-- seconds
--------------------------------------------------------
Time used for init cg fast fw in iter 5: --0.45-- seconds
----------------------------------------------------
Elapsed time: --inf-- seconds
----------------------------------------------------
CG relaxation obj. value in iter 0: 24.260951194847276
Time used for total init CG in iter 0: --10.73-- seconds
----------------------------------------------------
Elapsed time at CG iter 0: --inf-- seconds
----------------------------------------------------
======================================================
Column generation
Initial CG objective value: 24.260951194847276
CG iter        IA obj. value                 max slack value IA            sum slack values IA
1              24.260951194847276            0.96                          0.96
[1. 0. 1. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0. 1. 1. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0.56272047 0.43976428 0.46721342 0.28937013]
[0. 1. 0. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0.62094513 0.27034214 0.53394013 0.3348674 ]
[0.38414203 0.36013004 0.54796867 0.46747253]
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0.28886948 0.59156963 0.38188732 0.4963061 ]
CG iter        IA obj. value                 max slack value IA            sum slack values IA
2              24.260951194847276            0.96                          0.96
[1. 0. 1. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0. 1. 1. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0.56272047 0.43976428 0.46721342 0.28937013]
[0. 1. 0. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0.62094513 0.27034214 0.53394013 0.3348674 ]
[0.38414203 0.36013004 0.54796867 0.46747253]
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0.28886948 0.59156963 0.38188732 0.4963061 ]
CG iter        IA obj. value                 max slack value IA            sum slack values IA
3              24.260951194847276            0.96                          0.96
[1. 0. 1. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0. 1. 1. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0.56272047 0.43976428 0.46721342 0.28937013]
```

```
[0. 1. 0. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0.62094513 0.27034214 0.53394013 0.3348674 ]
[0.38414203 0.36013004 0.54796867 0.46747253]
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0.28886948 0.59156963 0.38188732 0.4963061 ]
CG iter        IA obj. value                max slack value IA           sum slack values IA
4              24.260951194847276           0.96                         0.96
[1. 0. 1. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0. 1. 1. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0.56272047 0.43976428 0.46721342 0.28937013]
[0. 1. 0. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0.62094513 0.27034214 0.53394013 0.3348674 ]
[0.38414203 0.36013004 0.54796867 0.46747253]
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0.28886948 0.59156963 0.38188732 0.4963061 ]
CG iter        IA obj. value                max slack value IA           sum slack values IA
5              24.260951194847276           0.96                         0.96
[1. 0. 1. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0. 1. 1. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0.56272047 0.43976428 0.46721342 0.28937013]
[0. 1. 0. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0.62094513 0.27034214 0.53394013 0.3348674 ]
[0.38414203 0.36013004 0.54796867 0.46747253]
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0.28886948 0.59156963 0.38188732 0.4963061 ]
Iteration limit
New columns added: [0, 0, 1, 0]
number of minlp subproblems solved during CG: 40
=======================================================
CG relaxation obj. value in iter 1: 24.260951194847276
Time used for CG: --5.81-- seconds
-------------------------------------------------
Elapsed time at CG iter 1: --inf-- seconds
-------------------------------------------------
Num of MINLP subproblems solved in iter loop *1*   40
Total number of minlp subproblems solved in iter 1: 60
Total number of columns in iter 1: 10
Columns in blocks in iter 1: [2, 2, 4, 2]
Time used for CG in iter 1: --5.81-- seconds
------------------------------------
CG regarding all blocks
[1. 0. 1. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0. 1. 1. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0.56272047 0.43976428 0.46721342 0.28937013]
[0. 1. 0. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
Time used for CG for all blocks: --0.67-- seconds
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
=======================================================
Find solution - projection from ia solution - local search
Solved Original MINLP TO problem with a resulting compliance of 13.59
Time used for FindSol in iter 1: --0.57-- seconds
-------------------------------------------------
Elapsed time at FindSol iter 1: --inf-- seconds
-------------------------------------------------
Total time used in iter 1: --7.05-- seconds
=======================================================
Column generation
Initial CG objective value: 24.260951194847276
CG iter        IA obj. value                max slack value IA           sum slack values IA
1              24.260951194847276           0.96                         0.96
[0.56272047 0.43976428 0.46721342 0.28937013]
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
CG iter        IA obj. value                max slack value IA           sum slack values IA
2              24.260951194847276           0.96                         0.96
[0.56272047 0.43976428 0.46721342 0.28937013]
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
```

```
CG iter        IA obj. value                 max slack value IA           sum slack values IA
3              24.260951194847276            0.96                         0.96
[0.56272047 0.43976428 0.46721342 0.28937013]
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
CG iter        IA obj. value                 max slack value IA           sum slack values IA
4              24.260951194847276            0.96                         0.96
[0.56272047 0.43976428 0.46721342 0.28937013]
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
CG iter        IA obj. value                 max slack value IA           sum slack values IA
5              24.260951194847276            0.96                         0.96
[0.56272047 0.43976428 0.46721342 0.28937013]
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
Iteration limit
New columns added: [0, 0, 0, 0]
number of minlp subproblems solved during CG: 10
=======================================================
CG relaxation obj. value in iter 2: 24.260951194847276
Time used for CG: --4.4-- seconds
-------------------------------------------------
Elapsed time at CG iter 2: --inf-- seconds
-------------------------------------------------
Num of MINLP subproblems solved in iter loop *2*   10
Total number of minlp subproblems solved in iter 2: 74
Total number of columns in iter 2: 10
Columns in blocks in iter 2: [2, 2, 4, 2]
Time used for CG in iter 2: --4.4-- seconds
------------------------------------
CG regarding all blocks
[1. 0. 1. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0. 1. 1. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0.56272047 0.43976428 0.46721342 0.28937013]
[0. 1. 0. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
Time used for CG for all blocks: --0.62-- seconds
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
=======================================================
Find solution - projection from ia solution - local search
Solved Original MINLP TO problem with a resulting compliance of 13.59
Could not find a better solution using start point provided by Decogo.
The compliance regarding the starting point equals 13.59
Time used for FindSol in iter 2: --0.62-- seconds
-------------------------------------------------
Elapsed time at FindSol iter 2: --inf-- seconds
-------------------------------------------------
Total time used in iter 2: --5.64-- seconds
=======================================================
Column generation
Initial CG objective value: 24.260951194847276
CG iter        IA obj. value                 max slack value IA           sum slack values IA
1              24.260951194847276            0.96                         0.96
[0.56272047 0.43976428 0.46721342 0.28937013]
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
CG iter        IA obj. value                 max slack value IA           sum slack values IA
2              24.260951194847276            0.96                         0.96
[0.56272047 0.43976428 0.46721342 0.28937013]
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
CG iter        IA obj. value                 max slack value IA           sum slack values IA
3              24.260951194847276            0.96                         0.96
[0.56272047 0.43976428 0.46721342 0.28937013]
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
CG iter        IA obj. value                 max slack value IA           sum slack values IA
4              24.260951194847276            0.96                         0.96
[0.56272047 0.43976428 0.46721342 0.28937013]
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
CG iter        IA obj. value                 max slack value IA           sum slack values IA
5              24.260951194847276            0.96                         0.96
[0.56272047 0.43976428 0.46721342 0.28937013]
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
Iteration limit
New columns added: [0, 0, 0, 0]
number of minlp subproblems solved during CG: 10
```

```
=======================================================
CG relaxation obj. value in iter 3: 24.260951194847276
Time used for CG: --4.63-- seconds
-------------------------------------------------
Elapsed time at CG iter 3: --inf-- seconds
-------------------------------------------------
Num of MINLP subproblems solved in iter loop *3*   10
Total number of minlp subproblems solved in iter 3: 88
Total number of columns in iter 3: 10
Columns in blocks in iter 3: [2, 2, 4, 2]
Time used for CG in iter 3: --4.63-- seconds
-----------------------------------
CG regarding all blocks
[1. 0. 1. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0. 1. 1. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0.56272047 0.43976428 0.46721342 0.28937013]
[0. 1. 0. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
Time used for CG for all blocks: --0.81-- seconds
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
=======================================================
Find solution - projection from ia solution - local search
Solved Original MINLP TO problem with a resulting compliance of 13.59
Could not find a better solution using start point provided by Decogo.
The compliance regarding the starting point equals 13.59
Time used for FindSol in iter 3: --0.62-- seconds
-------------------------------------------------
Elapsed time at FindSol iter 3: --inf-- seconds
-------------------------------------------------
Total time used in iter 3: --6.07-- seconds
=======================================================
Column generation
Initial CG objective value: 24.260951194847276
CG iter       IA obj. value                 max slack value IA           sum slack values IA
1             24.260951194847276            0.96                         0.96
[0.56272047 0.43976428 0.46721342 0.28937013]
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
CG iter       IA obj. value                 max slack value IA           sum slack values IA
2             24.260951194847276            0.96                         0.96
[0.56272047 0.43976428 0.46721342 0.28937013]
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
CG iter       IA obj. value                 max slack value IA           sum slack values IA
3             24.260951194847276            0.96                         0.96
[0.56272047 0.43976428 0.46721342 0.28937013]
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
CG iter       IA obj. value                 max slack value IA           sum slack values IA
4             24.260951194847276            0.96                         0.96
[0.56272047 0.43976428 0.46721342 0.28937013]
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
CG iter       IA obj. value                 max slack value IA           sum slack values IA
5             24.260951194847276            0.96                         0.96
[0.56272047 0.43976428 0.46721342 0.28937013]
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
Iteration limit
New columns added: [0, 0, 0, 0]
number of minlp subproblems solved during CG: 10
=======================================================
CG relaxation obj. value in iter 4: 24.260951194847276
Time used for CG: --4.44-- seconds
-------------------------------------------------
Elapsed time at CG iter 4: --inf-- seconds
-------------------------------------------------
Num of MINLP subproblems solved in iter loop *4*   10
Total number of minlp subproblems solved in iter 4: 102
Total number of columns in iter 4: 10
Columns in blocks in iter 4: [2, 2, 4, 2]
Time used for CG in iter 4: --4.44-- seconds
-----------------------------------
CG regarding all blocks
[1. 0. 1. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0. 1. 1. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0.56272047 0.43976428 0.46721342 0.28937013]
```

```
[0. 1. 0. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
Time used for CG for all blocks: --0.67-- seconds
----------------------------------------------------
Elapsed time: --inf-- seconds
----------------------------------------------------
======================================================
Find solution - projection from ia solution - local search
Solved Original MINLP TO problem with a resulting compliance of 13.59
Could not find a better solution using start point provided by Decogo.
The compliance regarding the starting point equals 13.59
Time used for FindSol in iter 4: --0.63-- seconds
------------------------------------------------
Elapsed time at FindSol iter 4: --inf-- seconds
------------------------------------------------
Total time used in iter 4: --5.74-- seconds
======================================================
Column generation
Initial CG objective value: 24.260951194847276
CG iter        IA obj. value                max slack value IA           sum slack values IA
1              24.260951194847276           0.96                         0.96
[0.56272047 0.43976428 0.46721342 0.28937013]
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
CG iter        IA obj. value                max slack value IA           sum slack values IA
2              24.260951194847276           0.96                         0.96
[0.56272047 0.43976428 0.46721342 0.28937013]
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
CG iter        IA obj. value                max slack value IA           sum slack values IA
3              24.260951194847276           0.96                         0.96
[0.56272047 0.43976428 0.46721342 0.28937013]
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
CG iter        IA obj. value                max slack value IA           sum slack values IA
4              24.260951194847276           0.96                         0.96
[0.56272047 0.43976428 0.46721342 0.28937013]
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
CG iter        IA obj. value                max slack value IA           sum slack values IA
5              24.260951194847276           0.96                         0.96
[0.56272047 0.43976428 0.46721342 0.28937013]
[1. 0. 0. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
Iteration limit
New columns added: [0, 0, 0, 0]
number of minlp subproblems solved during CG: 10
======================================================
CG relaxation obj. value in iter 5: 24.260951194847276
Time used for CG: --4.46-- seconds
------------------------------------------------
Elapsed time at CG iter 5: --inf-- seconds
------------------------------------------------
Num of MINLP subproblems solved in iter loop *5*   10
Total number of minlp subproblems solved in iter 5: 116
Total number of columns in iter 5: 10
Columns in blocks in iter 5: [2, 2, 4, 2]
Time used for CG in iter 5: --4.46-- seconds
------------------------------------
CG regarding all blocks
[1. 0. 1. 0.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0. 1. 1. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[0.56272047 0.43976428 0.46721342 0.28937013]
[0. 1. 0. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
Time used for CG for all blocks: --0.61-- seconds
----------------------------------------------------
Elapsed time: --inf-- seconds
----------------------------------------------------
======================================================
Find solution - projection from ia solution - local search
Solved Original MINLP TO problem with a resulting compliance of 13.59
Could not find a better solution using start point provided by Decogo.
The compliance regarding the starting point equals 13.59
Time used for FindSol in iter 5: --0.61-- seconds
------------------------------------------------
Elapsed time at FindSol iter 5: --inf-- seconds
------------------------------------------------
Total time used in iter 5: --5.68-- seconds
Iteration limit
Total time:                                 50.6037700176239
```

```
Reformulation time:                              0
Decomposition time:                              0
Containers time:                                 0
Primal bound:                                    13.5908647761
Main iterations:                                 5
Number of CG iterations:                         26
CG relaxation obj. value:                        24.260951194847276
Number of MINLP subproblems:                     120
Number of unfixed NLP subproblems:               28
Number of fixed NLP subproblems:                 0
Number of solved sub-problems after CG:          148
Number of columns after CG:                      10
CG Gap (CG relaxation and primal bound):         43.9804768369
Total number of columns:                         10
```

A.5.3 NLP Variant with Column Pertubation

```
Block separable reformulation:
Number of blocks:                                4
Number of nonlinear blocks:                      0
Min size of blocks:                              22
Max size of blocks (without linear blocks):      22
Max size of blocks (including linear blocks):    22
Number of vars:                                  88
Number of global constraints:                    23
Number of nonzero resources per block:           15,12,11,11
Number of equal./inequal. of global constraints:  22/1
-----------------------------------------------------
Used time: inf
-----------------------------------------------------
Initialization
Subgradient steps
Subgra.iter    Lagrange bound                alpha
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
1              84.86239504434809             1
[0.62094514 0.27034214 0.53394013 0.3348674 ]
[0.38414203 0.36013004 0.54796867 0.46747253]
[0.56272047 0.43976428 0.46721342 0.28937013]
[0.28886947 0.5915696  0.3818873  0.49630607]
2              45905.24143865322             1
[0.33985489 0.13998638 0.80795563 0.47149565]
[0.38414203 0.36013004 0.54796867 0.46747253]
[0.56272047 0.43976428 0.46721342 0.28937013]
[0.28886947 0.5915696  0.3818873  0.49630607]
3              2098643.653722888             2
[0.33985489 0.13998638 0.80795563 0.47149565]
[0.38414203 0.36013004 0.54796867 0.46747253]
[0.56272047 0.43976428 0.46721342 0.28937013]
[0.28886947 0.5915696  0.3818873  0.49630607]
4              2509751.9338034247            4
[0.33985489 0.13998638 0.80795563 0.47149565]
[0.38414203 0.36013004 0.54796867 0.46747253]
[0.56272047 0.43976428 0.46721342 0.28937013]
[0.28886947 0.5915696  0.3818873  0.49630607]
5              2993850.5211507073            8
Time used for SubGradient: --11.9-- seconds
-----------------------------------------------------
Elapsed time: inf
-----------------------------------------------------
=======================================================
Column generation: approximated subproblem solving
Initial CG objective value: 97.20469586403783
CG iter        IA obj. value                 max slack value IA            sum slack values IA
1              97.20469586403783             28.444273121373               285.77407535417353
[0.62094514 0.27034214 0.53394013 0.3348674 ]
Adding pertubated columns for Sub-Problems
[0.38414203 0.36013004 0.54796867 0.46747253]
Adding pertubated columns for Sub-Problems
[0.56272047 0.43976428 0.46721342 0.28937013]
Adding pertubated columns for Sub-Problems
[0.28886947 0.5915696  0.3818873  0.49630607]
Adding pertubated columns for Sub-Problems
[1. 0. 1. 0.]
Adding pertubated columns for Sub-Problems
[0. 0. 1. 1.]
Adding pertubated columns for Sub-Problems
[1. 1. 0. 0.]
Adding pertubated columns for Sub-Problems
```

```
[0.0705198  0.90029258 0.09322763 0.87703652]
Adding pertubated columns for Sub-Problems
Reduced costs greater than zero
New columns added: [0, 0, 0, 0]
number of minlp subproblems solved during CG: 0
=======================================================
Time used for init CG in iter 0: --104.16-- seconds
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
=======================================================
Find solution - init
Found approx. first primal bound of c=128.80
Pertubation Statistics: 0 of 10 points are infeasible
Time used for init FindSol in iter 0: --0.47-- seconds
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
Found the first feasible solution
IA obj. val: 175.44055579607996
Elapsed time: inf
---------------------------------------------------------
Fast column generation
iter      IA obj. value                 slacks
0         128.78137635521526            0.0
IA obj. val: 128.78137635521526
Elapsed time: inf
[0.54253359 0.3629303  0.49856094 0.40803825]
Adding pertubated columns for Sub-Problems
[0.49524748 0.48547023 0.59570901 0.54649562]
Adding pertubated columns for Sub-Problems
[0.42184549 0.29582884 0.32854706 0.23926326]
Adding pertubated columns for Sub-Problems
[0.371017   0.5444728  0.42139174 0.48264839]
Adding pertubated columns for Sub-Problems
No new columns generated in the current iteration
New columns in FastCG:
[0, 0, 0, 0]
number of unfixed nlp subproblems solved during CG: 4
Time used for solving subproblem: --15.43-- seconds
---------------------------------------------------------
Time used for init cg fast fw in iter 1: --15.86-- seconds
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
---------------------------------------------------------
Fast column generation
iter      IA obj. value                 slacks
0         121.60191308348365            0.0
IA obj. val: 121.60191308348365
Elapsed time: inf
[0.63383439 0.29049695 0.59863776 0.32660229]
Adding pertubated columns for Sub-Problems
[0.32686765 0.32104171 0.75875858 0.72940876]
Adding pertubated columns for Sub-Problems
[1.         0.16006721 0.         0.        ]
Adding pertubated columns for Sub-Problems
[0.21733338 0.73316245 0.24684177 0.69694711]
Adding pertubated columns for Sub-Problems
No new columns generated in the current iteration
New columns in FastCG:
[0, 0, 0, 0]
number of unfixed nlp subproblems solved during CG: 4
Time used for solving subproblem: --24.45-- seconds
---------------------------------------------------------
Time used for init cg fast fw in iter 2: --25.05-- seconds
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
---------------------------------------------------------
Fast column generation
iter      IA obj. value                 slacks
0         70.45301144523413             0.0
IA obj. val: 70.45301144523413
Elapsed time: inf
[0.85028292 0.11877783 0.83589179 0.13473593]
Adding pertubated columns for Sub-Problems
[0.10331136 0.10331136 1.         1.        ]
Adding pertubated columns for Sub-Problems
[0.46821659 0.46821659 0.         0.        ]
Adding pertubated columns for Sub-Problems
[0.08787543 0.89210831 0.0998067  0.87746519]
Adding pertubated columns for Sub-Problems
```

```
No new columns generated in the current iteration
New columns in FastCG:
[0, 0, 0, 0]
number of unfixed nlp subproblems solved during CG: 4
Time used for solving subproblem: --13.71-- seconds
---------------------------------------------------------
Time used for init cg fast fw in iter 3: --14.27-- seconds
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
---------------------------------------------------------
Fast column generation
iter      IA obj. value                slacks
0         64.63646908426908            0.0
IA obj. val: 64.63646908426908
Elapsed time: inf
[0.88215373 0.09349317 0.87082609 0.10511326]
Adding pertubated columns for Sub-Problems
[0.30569403 0.30569403 1.         1.        ]
Adding pertubated columns for Sub-Problems
[0.23851284 0.23851284 0.         0.        ]
Adding pertubated columns for Sub-Problems
[0. 1. 0. 1.]
Adding pertubated columns for Sub-Problems
No new columns generated in the current iteration
New columns in FastCG:
[0, 0, 0, 0]
number of unfixed nlp subproblems solved during CG: 4
Time used for solving subproblem: --111.8-- seconds
---------------------------------------------------------
Time used for init cg fast fw in iter 4: --112.37-- seconds
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
---------------------------------------------------------
Fast column generation
iter      IA obj. value                slacks
0         63.97977412742179            0.0
IA obj. val: 63.97977412742179
Elapsed time: inf
[0.8995618  0.0796825  0.88990747 0.08958609]
Adding pertubated columns for Sub-Problems
[0.39567443 0.39567443 1.         1.        ]
Adding pertubated columns for Sub-Problems
[0.14495664 0.14495664 0.         0.        ]
Adding pertubated columns for Sub-Problems
[0. 1. 0. 1.]
Adding pertubated columns for Sub-Problems
iter      IA obj. value                slacks
1         63.58039113952233            0.0
IA obj. val: 63.58039113952233
Elapsed time: inf
Number of new columns in the current iteration:
[0, 1, 0, 0]
[0.90994198 0.0714474  0.90128541 0.08032746]
Adding pertubated columns for Sub-Problems
[0.44229649 0.44229649 1.         1.        ]
Adding pertubated columns for Sub-Problems
[0.09620239 0.09620239 0.         0.        ]
Adding pertubated columns for Sub-Problems
[0. 1. 0. 1.]
Adding pertubated columns for Sub-Problems
No new columns generated in the current iteration
New columns in FastCG:
[0, 1, 0, 0]
number of unfixed nlp subproblems solved during CG: 8
Time used for solving subproblem: --243.46-- seconds
---------------------------------------------------------
Time used for init cg fast fw in iter 5: --244.06-- seconds
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
CG relaxation obj. value in iter 0: 63.58039113952233
Time used for total init CG in iter 0: --516.26-- seconds
-----------------------------------------------------
Elapsed time at CG iter 0: --inf-- seconds
-----------------------------------------------------
=====================================================
Column generation
Initial CG objective value: 60.58783924056511
CG iter        IA obj. value                max slack value IA            sum slack values IA
1              60.58783924056511            0.0                           0.0
[1. 0. 1. 0.]
```

```
[0. 0. 1. 1.]
[1. 0. 0. 0.]
[0. 1. 0. 1.]
Reduced costs greater than zero
New columns added: [0, 0, 0, 0]
number of minlp subproblems solved during CG: 4
=======================================================
CG relaxation obj. value in iter 1: 60.58783924056511
Time used for CG: --12.25-- seconds
-------------------------------------------------
Elapsed time at CG iter 1: --inf-- seconds
-------------------------------------------------
Num of MINLP subproblems solved in iter loop *1*   4
Total number of minlp subproblems solved in iter 1: 24
Total number of columns in iter 1: 2778
Columns in blocks in iter 1: [795, 795, 394, 794]
Time used for CG in iter 1: --12.25-- seconds
------------------------------------
CG regarding all blocks
[1. 0. 1. 0.]
[0. 0. 1. 1.]
[1. 0. 0. 0.]
[0. 1. 0. 1.]
Time used for CG for all blocks: --12.05-- seconds
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
CG converges, checking the convergence by exact subproblem solving
[1. 0. 1. 0.]
[0. 0. 1. 1.]
[1. 0. 0. 0.]
[0. 1. 0. 1.]
=======================================================
Find solution - projection from ia solution - local search
Time used for FindSol in iter 1: --0.98-- seconds
-------------------------------------------------
Elapsed time at FindSol iter 1: --inf-- seconds
-------------------------------------------------
Total time used in iter 1: --36.7-- seconds
CG converged
Total time:                                      564.8699369430542
Reformulation time:                              0
Decomposition time:                              0
Containers time:                                 0
Primal bound:                                    69.1573175395
Main iterations:                                 1
Number of CG iterations:                         2
CG relaxation obj. value:                        60.58783924056511
Number of MINLP subproblems:                     32
Number of unfixed NLP subproblems:               32
Number of fixed NLP subproblems:                 0
Number of solved sub-problems after CG:          64
Number of columns after CG:                      2778
CG Gap (CG relaxation and primal bound):         12.3912802936
Total number of columns:                         2778
```

A.5.4 MINLP Variant with Column Pertubation

```
Block separable reformulation:
Number of blocks:                                4
Number of nonlinear blocks:                      0
Min size of blocks:                              22
Max size of blocks (without linear blocks):      22
Max size of blocks (including linear blocks):    22
Number of vars:                                  88
Number of global constraints:                    23
Number of nonzero resources per block:           15,12,11,11
Number of equal./inequal. of global constraints: 22/1
-----------------------------------------------------
Used time: inf
-----------------------------------------------------
Initialization
Subgradient steps
Subgra.iter    Lagrange bound               alpha
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
[0.44 0.44 0.44 0.44]
1              86.05485233539662            1
[0.61305593 0.2842367  0.52188807 0.34136498]
[0.38414203 0.36013004 0.54796869 0.46747254]
```

```
[0.56272047 0.43976426 0.46721343 0.28937012]
[0.2888695  0.59156965 0.38188734 0.49630612]
2            61909.295921364566          1
[0.61305593 0.2842367  0.52188807 0.34136498]
[0.38414203 0.36013004 0.54796869 0.46747254]
[0.56272047 0.43976426 0.46721343 0.28937012]
[0.2888695  0.59156965 0.38188734 0.49630612]
3            5933971.901724353           2
[0.61305593 0.2842367  0.52188807 0.34136498]
[0.38414203 0.36013004 0.54796869 0.46747254]
[0.56272047 0.43976426 0.46721343 0.28937012]
[0.2888695  0.59156965 0.38188734 0.49630612]
4            6856331.40950469            4
[0.61305593 0.2842367  0.52188807 0.34136498]
[0.38414203 0.36013004 0.54796869 0.46747254]
[0.56272047 0.43976426 0.46721343 0.28937012]
[0.2888695  0.59156965 0.38188734 0.49630612]
5            5990543.492285435           8
Time used for SubGradient: --4.63-- seconds
-----------------------------------------------------
Elapsed time: inf
-----------------------------------------------------
=======================================================
Column generation: approximated subproblem solving
Initial CG objective value: 107.25801395263895
CG iter      IA obj. value               max slack value IA          sum slack values IA
1            107.25801395263895          24.59411312627706           154.10823067785853
[0.61305593 0.2842367  0.52188807 0.34136498]
Adding pertubated point for Sub-Problems
[0.38414203 0.36013004 0.54796869 0.46747254]
Adding pertubated point for Sub-Problems
[0.56645269 0.43906377 0.46608543 0.28746896]
Adding pertubated point for Sub-Problems
[0.2888695  0.59156965 0.38188734 0.49630612]
Adding pertubated point for Sub-Problems
[0.61351114 0.28390232 0.52245054 0.34213982]
Adding pertubated point for Sub-Problems
[0.39739267 0.37389732 0.55769448 0.45741452]
Adding pertubated point for Sub-Problems
[0.59961351 0.48703105 0.42779488 0.26495612]
Adding pertubated point for Sub-Problems
[0.2888695  0.59156965 0.38188734 0.49630612]
Adding pertubated point for Sub-Problems
Reduced costs greater than zero
New columns added: [0, 0, 0, 0]
number of minlp subproblems solved during CG: 0
=======================================================
Time used for init CG in iter 0: --21.58-- seconds
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
=======================================================
Find solution - init
Attention: Updated r.h.s. of volume constraint to 8.0 due to MINLP solving.
Found approx. first primal bound of MINLP with c=24.260951194847255
Found approx. first primal bound of c=24.26
Pertubation Statistics: 0 of 10 points are infeasible
Time used for init FindSol in iter 0: --0.41-- seconds
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
Found the first feasible solution
IA obj. val: 79.87997697515975
Elapsed time: inf
---------------------------------------------------------
Fast column generation
iter     IA obj. value               slacks
0        24.260951194847276          0.96
[1. 0. 1. 0.]
Adding pertubated point for Sub-Problems
[0. 1. 1. 1.]
Adding pertubated point for Sub-Problems
[1. 0. 0. 0.]
Adding pertubated point for Sub-Problems
[0. 1. 0. 1.]
Adding pertubated point for Sub-Problems
No new columns generated in the current iteration
New columns in FastCG:
[0, 0, 0, 0]
number of unfixed nlp subproblems solved during CG: 4
Time used for solving subproblem: --8.62-- seconds
---------------------------------------------------------
Time used for init cg fast fw in iter 1: --9.01-- seconds
```

```
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
---------------------------------------------------------
Fast column generation
iter      IA obj. value                 slacks
0         24.260951194847276            0.0
IA obj. val: 24.260951194847276
Elapsed time: inf
[1. 0. 1. 0.]
Adding pertubated point for Sub-Problems
[0. 1. 1. 1.]
Adding pertubated point for Sub-Problems
[0. 0. 0. 0.]
Adding pertubated point for Sub-Problems
[0. 1. 0. 1.]
Adding pertubated point for Sub-Problems
No new columns generated in the current iteration
New columns in FastCG:
[0, 0, 0, 0]
number of unfixed nlp subproblems solved during CG: 4
Time used for solving subproblem: --8.73-- seconds
---------------------------------------------------------
Time used for init cg fast fw in iter 2: --9.11-- seconds
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
---------------------------------------------------------
Fast column generation
iter      IA obj. value                 slacks
0         24.260951194847276            0.0
IA obj. val: 24.260951194847276
Elapsed time: inf
[1. 0. 1. 0.]
Adding pertubated point for Sub-Problems
[0. 1. 1. 1.]
Adding pertubated point for Sub-Problems
[0. 0. 0. 0.]
Adding pertubated point for Sub-Problems
[0. 1. 0. 1.]
Adding pertubated point for Sub-Problems
iter      IA obj. value                 slacks
1         24.260951194847276            0.0
IA obj. val: 24.260951194847276
Elapsed time: inf
Number of new columns in the current iteration:
[0, 0, 1, 0]
[1. 0. 1. 0.]
Adding pertubated point for Sub-Problems
[0. 1. 1. 1.]
Adding pertubated point for Sub-Problems
[0. 0. 0. 0.]
Adding pertubated point for Sub-Problems
[0. 1. 0. 1.]
Adding pertubated point for Sub-Problems
No new columns generated in the current iteration
New columns in FastCG:
[0, 0, 1, 0]
number of unfixed nlp subproblems solved during CG: 8
Time used for solving subproblem: --30.69-- seconds
---------------------------------------------------------
Time used for init cg fast fw in iter 3: --31.09-- seconds
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
---------------------------------------------------------
Fast column generation
iter      IA obj. value                 slacks
0         24.260951194847276            0.0
IA obj. val: 24.260951194847276
Elapsed time: inf
[1. 0. 1. 0.]
Adding pertubated point for Sub-Problems
[0. 1. 1. 1.]
Adding pertubated point for Sub-Problems
[0. 0. 0. 0.]
Adding pertubated point for Sub-Problems
[0. 1. 0. 1.]
Adding pertubated point for Sub-Problems
No new columns generated in the current iteration
New columns in FastCG:
[0, 0, 0, 0]
number of unfixed nlp subproblems solved during CG: 4
```

```
Time used for solving subproblem: --9.68-- seconds
---------------------------------------------------------
Time used for init cg fast fw in iter 4: --10.12-- seconds
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
---------------------------------------------------------
Fast column generation
iter      IA obj. value                 slacks
0         24.260951194847276            0.0
IA obj. val: 24.260951194847276
Elapsed time: inf
[1. 0. 1. 0.]
Adding pertubated point for Sub-Problems
[0. 1. 1. 1.]
Adding pertubated point for Sub-Problems
[0. 0. 0. 0.]
Adding pertubated point for Sub-Problems
[0. 1. 0. 1.]
Adding pertubated point for Sub-Problems
No new columns generated in the current iteration
New columns in FastCG:
[0, 0, 0, 0]
number of unfixed nlp subproblems solved during CG: 4
Time used for solving subproblem: --7.07-- seconds
---------------------------------------------------------
Time used for init cg fast fw in iter 5: --7.52-- seconds
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
CG relaxation obj. value in iter 0: 24.260951194847276
Time used for total init CG in iter 0: --88.85-- seconds
-----------------------------------------------------
Elapsed time at CG iter 0: --inf-- seconds
-----------------------------------------------------
=======================================================
Column generation
Initial CG objective value: 24.260951194847276
CG iter        IA obj. value                 max slack value IA            sum slack values IA
1              24.260951194847276            0.0                           0.0
[1. 1. 1. 1.]
C:\FinalThesis\DecogoTOLayer\pylib\simp\simp.py:135: RuntimeWarning: overflow encountered in true_divide
  np.minimum(1.0, np.minimum(x_old + move, x_old * (-dc / lmid) ** eta))))
C:\FinalThesis\DecogoTOLayer\pylib\simp\simp.py:135: RuntimeWarning: divide by zero encountered in true_divide
  np.minimum(1.0, np.minimum(x_old + move, x_old * (-dc / lmid) ** eta))))
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[1. 1. 1. 0.]
[1. 0. 1. 0.]
[0. 1. 1. 1.]
Reduced costs greater than zero
New columns added: [0, 0, 0, 0]
number of minlp subproblems solved during CG: 4
=======================================================
CG relaxation obj. value in iter 1: 24.260951194847276
Time used for CG: --1.07-- seconds
-------------------------------------------------
Elapsed time at CG iter 1: --inf-- seconds
-------------------------------------------------
Num of MINLP subproblems solved in iter loop *1*   4
Total number of minlp subproblems solved in iter 1: 24
Total number of columns in iter 1: 1286
Columns in blocks in iter 1: [394, 394, 104, 394]
Time used for CG in iter 1: --1.07-- seconds
-------------------------------------
CG regarding all blocks
[1. 1. 1. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[1. 1. 1. 0.]
[1. 0. 1. 0.]
[0. 1. 1. 1.]
Time used for CG for all blocks: --1.05-- seconds
-----------------------------------------------------
Elapsed time: --inf-- seconds
-----------------------------------------------------
CG converges, checking the convergence by exact subproblem solving
[1. 1. 1. 1.]
SIMP-SubProblem is not solved! There were illogical initial values provided by Decogo!
[1. 1. 1. 0.]
[1. 0. 1. 0.]
[0. 1. 1. 1.]
=======================================================
Find solution - projection from ia solution - local search
C:\FinalThesis\DecogoTOLayer\pylib\simp\simp.py:135: RuntimeWarning: invalid value encountered in sqrt
```

```
  np.minimum(1.0, np.minimum(x_old + move, x_old * (-dc / lmid) ** eta))))
C:\FinalThesis\DecogoTOLayer\pylib\simp\simp.py:135: RuntimeWarning: invalid value encountered in multiply
  np.minimum(1.0, np.minimum(x_old + move, x_old * (-dc / lmid) ** eta))))
C:\FinalThesis\DecogoTOLayer\pylib\simp\simp.py:135: RuntimeWarning: invalid value encountered in true_divide
  np.minimum(1.0, np.minimum(x_old + move, x_old * (-dc / lmid) ** eta))))
Can not solve original SIMP-Problem due to presented initial values!
Time used for FindSol in iter 1: --0.51-- seconds
-------------------------------------------------
Elapsed time at FindSol iter 1: --inf-- seconds
-------------------------------------------------
Total time used in iter 1: --3.22-- seconds
CG converged
Total time:                                 96.69659757614136
Reformulation time:                         0
Decomposition time:                         0
Containers time:                            0
Primal bound:                               24.2609511948
Main iterations:                            1
Number of CG iterations:                    2
CG relaxation obj. value:                   24.260951194847276
Number of MINLP subproblems:                32
Number of unfixed NLP subproblems:          32
Number of fixed NLP subproblems:            0
Number of solved sub-problems after CG:     64
Number of columns after CG:                 1286
CG Gap (CG relaxation and primal bound):    0.0
Total number of columns:                    1286
```

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen.“

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Dieses Blatt, mit der folgenden Erklärung, ist nach Fertigstellung der Abschlussarbeit durch den Studierenden auszufüllen und jeweils mit Originalunterschrift als letztes Blatt in das Prüfungsexemplar der Abschlussarbeit einzubinden.
Eine unrichtig abgegebene Erklärung kann -auch nachträglich- zur Ungültigkeit des Studienabschlusses führen.

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: Mau

Vorname: Sebastian

dass ich die vorliegende Masterarbeit bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Topology Optimization using Column Generation Methods

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

- die folgende Aussage ist bei Gruppenarbeiten auszufüllen und entfällt bei Einzelarbeiten -

Die Kennzeichnung der von mir erstellten und verantworteten Teile der Masterarbeit ist erfolgt durch:

Hamburg, den	14.03.2022	
Ort	Datum	Unterschrift im Original